AN INVESTIGATION OF ERROR CHARACTERISTICS AND CODING PERFORMANCE

NASA GRANT NAG5-2006

Annual Report
January 1993 - August 1993

Submitted to:

Mr. Warner Miller
Code 728.4
Instrument Electronics Systems Branch
Engineering Directorate
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
(301)+286-8183

Submitted by:

William J. Ebel, Ph.D.
Frank M. Ingels, Ph.D.
Mississippi State University
Drawer EE
Mississippi State, MS 39762
(601)+325-3912

August 1993

1.0 INTRODUCTION

The first year's effort on NASA Grant NAGS-2006 was an investigation to characterize typical errors resulting from the EOS dorn link. The analysis methods developed for this effort were used on test data from a March 1992 White Sands Terminal Test.

The effectiveness of a concatenated coding scheme of a Reed Solomon outer code and a convolutional inner code versus a Reed Solomon only code scheme has been investigated as well as the effectiveness of a Periodic Convolutional Interleaver in dispersing errors of certain types.

The work effort consisted of development of software that allows simulation studies with the appropriate coding schemes plus either simulated data with errors or actual data with errors. The annual report dated July 1, 1992 to December 30, 1992, Appendix I, reports in detail on the software tools developed and delivered to NASA/GSFC. The software program is entitled Communication Link Error Analysis (CLEAN) and models downlink errors, forward error correcting schemes, and interleavers.

The quarterly report, January 1993 to March 30, 1993, Appendix II, details the analysis of the real error sequence from the Ku-band downlink obtained from NASA/GSFC in February 1993. This error stream was analyzed using statistical tests. The Ku-band downlink errors through the TDRS to White Sands link are shown to be random errors. This conclusion is drawn by (1) comparing the empirical error interval distribution from the real EOS data to the theoretical error interval distribution assuming random occurrences of error pairs at the NRZM decoder output and (2) performing the CVM (CRAMER VON-MISES) distribution test on a subset of the real data. In this quarterly report, a study of the performance of the (255, 223) Reed-Solomon (RS) code on bursty errors is also present as well as a brief study dealing with synchronization for the CCSDS transfer frame format which includes a 32 bit PN sequence header.

The report in Appendix III report gives a summary of the work from July 1, 1992 to June 30, 1993 as well as giving a set of curves comparing various coding scheme performances as determined via CLEAN for NRZM decoding. These curves are presented for RS Coding, Convolutional Coding, DPCI/Convolutional Coding, Concatenated Coding, RS Coding with erasure decoding. These codes were compared for decoded bit error rate performance for Random-Error Channels and Bursty-Error Channels. The Bursty Error Channels are analyzed for different mixes of bursts versus random errors.

Comments on Rice Compression pitfalls is also included in the Appendix III material.

APPENDIX I

An Investigation of Error Characteristics and Coding Performance

NASA GRANT NAG5-2006

Annual Report

January 1993 - August 1993

An Investigation of
Error Characteristics and Coding Performance

NASA GRANT NAG5-2006
July 1, 1992 - June 30, 1993

Semi-Annual Report 1
July 1, 1992 - Dec 30, 1992

Submitted to:

Mr. Warner Miller
Code 728.4
Instrument Electronic Systems Branch
Engineering Directorate
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
301-286-8183

Submitted by:

William J. Ebel, Ph.D.
Frank M. Ingels, Ph.D.
Mississippi State University
Drawer EE
Mississippi State, MS 39762
601-325-3912

December 1992

I. INTRODUCTION

This report describes research performed to date on NASA Grant NAG5-2006 for the period July 1, 1992 through December 1, 1992. This work involves studying the performance of forward error correcting coding schemes on errors anticipated for the Earth Observation System (EOS) Ku-band downlink.

The EOS transmits picture frame data to the ground via the Telemetry Data Relay Satellite System (TDRSS) to a ground-based receiver at White Sands. Due to unintentional RF interference from other systems operating in the Ku band, the noise at the receiver is non-Gaussian which may result in non-random errors output by the demodulator. That is, the downlink channel cannot be modeled by a simple memoryless Gaussian-noise channel. From previous experience, it is believed that those errors are bursty.

The research has proceeded by developing a computer based simulation, called Communication Link Error ANalysis (CLEAN), to model the downlink errors, forward error correcting schemes, and interleavers used with TDRSS. To date, the bulk of CLEAN, described in Sections 3, 4, and 5, has been written, documented, debugged, and verified. The procedures for utilizing CLEAN to investigate code performance have been established and will be discussed in Section 5.

II. SOURCE CODE GENERAL DESCRIPTION

Each system component (decoder, deinterleaver, etc.) has been implemented in CLEAN as separate executable computer programs which interface with each other through data files including an error sequence data file. This allows them to be executed sequentially via a batch file.

All computer programs read parameters from a separate ASCII parameter file with a fixed default name. The default name for the parameter file is the same as the executable but has the extension 'prm'. Also, there is a global parameter file, 'ID.prm', which contains a simulation identifier (ID). Each program generates an output file with an extension identical to this ID. This output file contains all the calculated statistics and estimated parameters from the program. This allows all the files generated by a specific run to be quickly identified and distinguished from data files generated by other runs.

To conduct the studies, a batch file is created which contains a series of executable programs. The type and order of the executables in the batch file implements a particular system configuration. For example, if the user chooses to use a Reed-Solomon (RS) decoder to decode a sequence of random errors, then the batch file contains two executables; the first generates a random error sequence and the second uses an RS decoder to correct them. In general, the batch file contains one of the channel error sequence generation programs which will generate an error sequence stored in file name 'error.seq'. Each program which is executed makes use of and/or modifies that error sequence and generates statistics and other outputs for the error pattern.

The programs have been written with parameter bounds in mind. For example, the programs are designed so that the lowest channel average error probability to be investigated, coded or uncoded, is roughly 10^{-6}. Along with this, it is assumed that 20 errors are the minimum number required to characterize the statistics of the channel, however, in general many more errors will be generated per sequence. Thus as an upper bound, generating an error sequence, coded or uncoded, with an error probability of 10^{-6} requires a minimum of $20/10^{-6} = 2\text{x}10^{7}$ error sequence values. The error sequence file is stored in a "packed" format so that 15 error sequence values are stored per two bytes of memory. Therefore, the largest error sequence file is $2\text{x}10^{7}/(2/15) =$ 2.67Mbytes. This is sufficiently small so that allowable disc space on most computers can accommodate several files at once. In general, error files are ***not*** stored but are generated on the fly. Results can be reproduced by regenerating an error sequence given the proper random number generator and the seed. If it turns out that regenerating the error sequence takes too long, then a set of error sequences can be generated and stored on disc or magnetic tape to be retrieved when required.

All programs have been documented upon completion with a documentation test run. All the generated documentation is stored in a common binder for later reference.

Each program conforms to a documentation standard which includes a program/subroutine/function header as well as line comments within the code. On average, there should be a comment line per 6 lines of code to indicate the purpose of the next few lines of code. The routine header takes the following form:

```
c*********************************************************************
c*
c* - Program/Subroutine/Function name:  name (Acronym meaning)
c*
c* - Purpose:  This program/subroutine/function ...
c*
c*
c* - Revision History:
c*         Date          Who                     Reason
c*    -----------------------------------------------------------------
c*      May 25, 1992      WE                    Original
c*
c*
c* - Variable/File List:
c*           Name     Type              Description
c*    ------------------------------------------------------------------
c*     Inputs:
c*
c*     Outputs:
c*
c*     Internals:
c*
c*
c* - Subroutines called:
c* - Subroutines called by:
c* - Functions called:
c* - Functions called by:
c*
c***********************************************************************
```

As an example, a program written to create a bursty-error sequence may have a header which appears as follows:

```
c*********************************************************************
c*
c* - Program name:  BstyErrS (Bursty-Error Sequence)
c*
c* - Purpose:  This program generates an binary error sequence with
c*     bursty errors.  The error sequence denotes a correct binary channel
c*     transmission with a 0 and denotes an error with a 1.  The error
c*     sequence is partitioned into two main, noncontiguous parts, the burst
c*     error part and the thermal error part.  The method used to generate
c*     each part of the error sequence depends upon the density of errors to
c*     be generated.  For each error sequence part, if the required density of
c*     errors is greater than .01, then the program uses a conditional test on
c*     a uniform random number in the range [0,1]. If the density of errors is
c*     less than .01, then the program will use a sample from the exponential
c*     distribution to generate the next error occurrence time.
c*       This program inputs parameters from an ASCII data file with default
c*     name 'BstyErrs.prm' and outputs the error sequence to a data file
c*     with default name 'error.seq'.  In addition, various statistics are
c*     output to an ASCII data file with default name 'BstyErrs.ID', where
c*     ID is a three letter identifier for the current run which is input from
c*     file 'ID.prm'.
c*       The program is run by editing the parameter file 'BstyErrs.prm' and
c*     selecting the appropriate parameters and by choosing a program ID by
c*     editing file 'ID.prm'.  Executing the program generates the 'error.seq'
c*     file which contains an error sequence (in packed format) with
c*     binomially distributed errors.  It does not matter whether the output
c*     file 'error.seq' exists or not. If it exists, it is overwritten without
c*     a prompt to the user.
c*       Even though Poisson distributed bursts may overlap in theory, this
c*     progam does not allow error bursts to overlap.  The user must take care
c*     to specify input parameters so that the probability of overlapping
c*     burst is negligible.  It is also assumed that Peg<Peb.
```

```
c*
c*
c* - Revision History:
c*         Date            Who                       Reason
c*    ----------------------------------------------------------------
c*      Aug 20, 1992      WE                        Original
c*     Sept 14, 1992      WE       Modified to use Makefile to link source
c*                                 and updated the documentation
c*      Oct  2, 1992      WE       Output Number of Errors to the error.seq
c*                                 file header
c*      Nov  4, 1992      WE       Updated NextBurst function argument list
c*                                 to include the previous burst length
c*      Nov 13, 1992      WE       Added write to output Log10(Density)
c*      Nov 16, 1992      WE       Changed all real variables to double precision
c*
c*
c* - Variable/File List:
c*     Inputs:  None (See subroutine ReadParams)
c*
c*     Outputs:
c*          Name       Type              Description
c*      ----------------------------------------------------------------
c*       error.seq    file         Error sequence output file
c*                                 (in packed format)
c*          Nerrs    integer*4     Total Number of errors generated
c*      ErrDensity    real*8       Total Error density for generated seq
c*      NBurstyErrs integer*4      Number of errors in the bursts
c*      GenBurstDen   real*8       Error density within the error bursts
c*      GenThermDen   real*8       Error density outside the error bursts
c*         NBursts   integer*4     Total number of bursts generated
c*     GenMeanIntv    real*8       Average burst occurrence
c*     TotalBLength  integer*4     Total sum of burst lengths
c*     GenBDuration   real*8       Average burst length (seq sym)
c*
c*     Internals:
c*           ID     character*3    Identifier for statistics output file
c*            N      integer*4     Error sequence length
c*           Tbs      real*8       Binary channel symbol frequency (freq.)
c*           Peg      real*8       Thermal error density
c*         PegSeed    real*8       Peg random number generator seed
c*           Peb      real*8       Burst error density
c*         PebSeed    real*8       Peb random number generator seed
c*         IntvFlag integer*4      = 1, Periodic error occurrence times
c*                                 = 2, Gaussian error occurrence times
c*                                 = 3, Poisson error occurrence times
c*         IntvMean integer*4      Burst occurrence rate (interval mean)
c*         IntvSeed   real*8       Interval random number generator seed
c*         IntvVar  integer*4      Burst occurrence rate variance
c*                                 (interval statistic variance)
c*        LngthFlag integer*4      = 1, Fixed length error bursts
c*                                 = 2, Gaussian dist. error burst lengths
c*                                 = 3, Exponential error burst lengths
c*        LngthMean integer*4      Burst length distribution mean
c*        LngthSeed   real*8       Length random number generator seed
c*        LngthVar  integer*4      Burst length distribution variance
c*           i,j     integer*4     Do loop indices
c*         RecNum    integer*4     Record number index (error.seq file)
c*         NseqSym   integer*4     Number of DBESS
c*        Error(15)  integer*4     Contains 15 error sequence values
c*           zero    integer*4     Identically the number 0
c*   BurstIntvCount  integer*4     Interval Count to next error burst
c*       PrevLength  integer*4     Previous Burst Length
c*  ErrorBurstCount  integer*4     Length of next error burst (seq sym)
c*     PegIntvCount  integer*4     Interval Count to next Therm error
c*     PebIntvCount  integer*4     Interval Count to next burst error
c*          DBESS    integer*4     15 consecutive error sequence values
c*                                 stored in a 2 byte integer. Stands
c*                                 for Double Byte Error Sequence Symbol
c*           URV      real*8       Uniform random variable in [0,1]
c*       NSplit(2)   integer*2     A dummy array used to access each
c*                                 double byte of the integer*4
c*                                 number N.
c*      NESplit(2)   integer*2     A dummy array used to access each
c*                                 double byte of the integer*4
c*                                 number Nerrs.
c*
c*
c* - Subroutines called: ReadParams, IterBinErrGen
c* - Functions called: PackErrors, UniformRV, NextBurst, NextLength
c*
c**********************************************************************
```

Figure 1 shows an overall block diagram depicting the CLEAN simulation capability. The CLEAN simulation requires the following assumptions:

1) The transmitted data is all zero

2) Synchronization has been established (i.e. only steady state error statistics are considered)

3) Demodulator performs hard decisions

At each of the points labeled A, B, C, D, and E shown in Figure 1, it is possible to perform statistical analysis including (see Section III.D below):

1) Perform the Cramer Von Mises distribution test to determine if the errors are random.

2) Perform the Cramer Von Mises distribution test on blocks of the error sequence.

3) Estimate burst-error parameters

   a) Average burst-error length
   b) Variance of the burst-error length
   c) List of the burst-error lengths
   d) Average random interval length
   e) Variance of the random interval length
   d) List of the random interval lengths

4) The error interval histogram (for random errors this should be an exponential distribution)

5) Determination of the burst-error distribution 'ala' CLASS

For each program, the calculated statistics are output to the log file as described above.

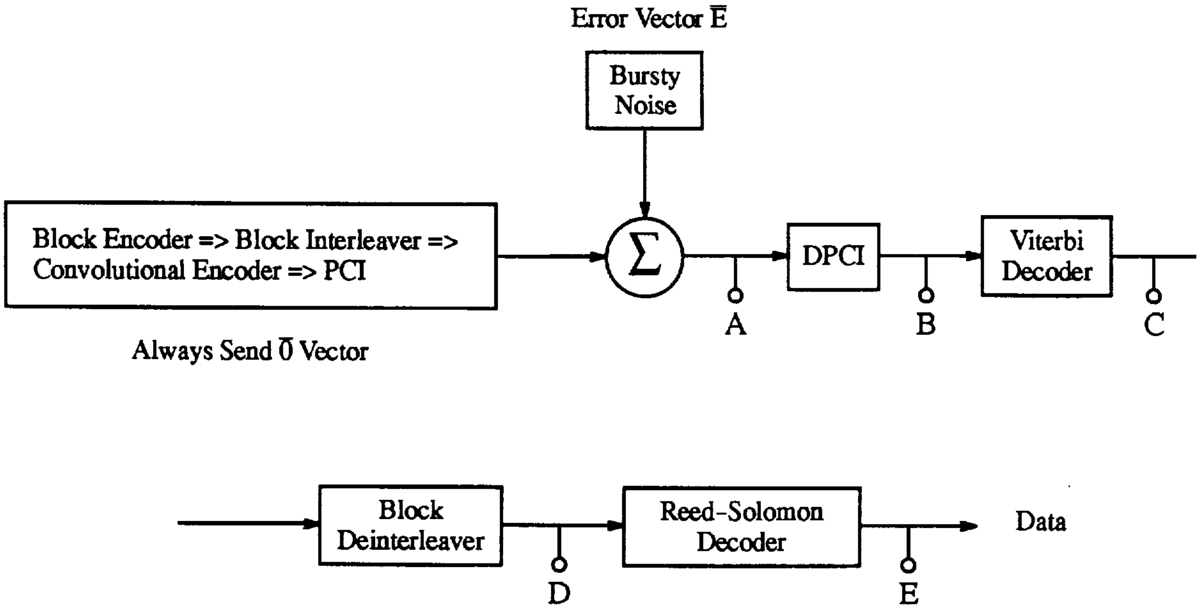


Figure 1. Overall block diagram depicting the CLEAN simulation capability.

III. PROGRAM DESCRIPTIONS

In this section, the programs which deal with the TDRSS system simulation are briefly described.

A. Forward Error Correcting Codes

The contract requires that Reed-Solomon codes and convolutional codes be considered. Reed-Solomon codes are a class of block codes. To this end, a program is described which implements the effect of an (n,k,m,t) block incomplete, errors-only decoder and a separate program to implement a Viterbi decoder which is used to decode convolutional codes.

1. BlkDecod (Block Decoder)

This program performs the effect of an incomplete, errors (erasure) only decoder. The program operates by simply partitioning the error sequence into blocks equivalent to a received codeword. Error statistics are calculated from each block including the number of bit errors and the number of code symbol errors. If the incomplete decoder detects more errors than the error correcting capability of the code, then the errors are not corrected, otherwise they are.

This program inputs parameters from an ASCII data file with default name 'BlkDecod.prm' and inputs the error sequence from the file with default name 'error.seq'. The decoded error sequence is output to the 'error.seq' file and various statistics are output to an ASCII data file with default name 'BlkDecod.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'BlkDecod.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program generates the 'error.seq' file which contains an error sequence (in packed format) with decoded errors. *The 'error.seq' file must exist prior to the execution of this program.*

There is one important assumption associated with the output of this program. It is assumed that the undetected word error probability is negligible. This is important because this program does not implement an actual decoding algorithm, rather the decoded error sequence is constructed by simply counting errors. Under certain circumstances, it is possible for the errors to occur in such a way so that the received codeword is mapped to within a sphere of t (error correcting capability of the code) about the wrong codeword. A decoding algorithm *cannot* detect (all by itself) that error pattern because it thinks that only a few errors occurred which are then corrected to the wrong codeword. The probability that this event occurs is called the undetected word error probability. The algorithm implemented here cannot tell whether an error pattern is undetectable by a true decoding algorithm. Therefore, this probability is assumed to be negligible which is, in general, a valid assumption.

2. Viterbi

This program performs hard decision Viterbi decoding assuming the all zero sequence is transmitted. The Viterbi decoding algorithm assumes that the trellis begins at the all zero state for the first received code symbol. The end of the decoding process does not terminate with flush bits. Instead, steady state Viterbi decoding is performed up to the end of the error sequence.

This program inputs parameters from an ASCII data file with default name 'Viterbi.prm' and outputs the decoded error sequence to data file with default name 'error.seq'. In addition, various statistics are output to an ASCII data file with default name 'Viterbi.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'Viterbi.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program generates the 'error.seq' file which contains an error sequence (in packed format) with the decoded error sequence. *The 'error.seq' file must exist prior to the execution of this program.* There are several assumptions associated with the implementation and output of this program.

1) It is assumed that the all zero sequence is transmitted,

2) The path with the minimum Hamming distance at the i^{th} Trellis stage is used to find the decoded bit for the output,

3) It is assumed that the convolutional encoder is either rate 1/2 or rate 1/3. It is straight forward to extrapolate this program to accommodate a rate $1/n$ encoder, however this has not been done to date. It should also be possible to modify this program to accommodate a rate m/n encoder using the concept of a punctured convolutional code, again however, this has not been done to date.

The Viterbi algorithm, as implemented here, updates the Trellis by iterating through each of the states at the next stage. The Hamming distance for each path entering a given state are computed and the survivor is kept while the other sequence is discarded. In case of a tie, a coin is flipped (via a Uniform RV in [0,1]) to determine the survivor. The survivor is identified by updating the MLStateTrace array. This array contains the state of the previous Trellis stage which connects to the given state being processed. For example, suppose that we are now processing the next stage in the Trellis, we first consider state 1 at the next stage. After investigating the Hamming distances for the two possible paths entering state 1, we find that the survivor path came from state 3 of the previous Trellis stage. Therefore, MLStateTrace(i,1) = 3 where i is the stage index.

To prevent overwriting the Metric array, two Metric arrays are alternately processed for each Trellis stage. This is why the algorithm performs two Trellis stage updates for each main loop. In the first Trellis stage update, the metrics are found in array MetricA and the new metrics are stored in MetricB. In the second Trellis stage update, the metrics are found in array MetricB and the new metrics are stored in MetricA.

The Trellis is defined via three arrays; PathCodeSym, PathLink, and PathBit. Since this program only accommodates rate 1/2 or 1/3 encoders, only two paths enter each state at a given trellis stage. therefore, if there are N trellis states, then there are only 2*N possible paths between two trellis stages. These are sequentially numbered from 1 to 2*N where path number 1 and 2 enter state 1, path 3 and 4 enter state 2, etc. Array PathLink(i) gives the state number from which path i originates. Also, PathCodeSym(i) gives the code symbol associated with path i, and PathBit(i) gives the bit associated with path i. Taken together, these three arrays completely define the steady state trellis.

B. Channel Error Sequences

The contract requires that several types of channel errors be considered. A program is described which generates Binomial (random) errors which would occur if the channel noise was additive white Gaussian noise (AWGN). Two other programs are described which generate burst errors and bursty errors. These allow the error bursts to have a variety of length statistics and occurrence statistics in addition to a variety of error density statistics.

1. BinErrs (Binomial Error Sequence generation)

This program generates an binary error sequence with binomially distributed errors. The error sequence denotes a correct binary channel transmission with a 0 and denotes an error with a 1. The method used to generate the error sequence depends upon the density of errors to be generated. If the required density of errors is greater than 0.01, then the program uses a conditional test on a uniform random number in the range [0,1]. If the density of errors is less than 0.01, then the program uses a sample from the exponential distribution to generate the next error occurrence time.

This program inputs parameters from an ASCII data file with default name 'BinErrs.prm' and outputs the error sequence to data file with default name 'error.seq'. In addition, various statistics are output to an ASCII data file with default name 'BinErrs.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'BinErrs.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program generates the 'error.seq' file which contains an error sequence (in packed format) with binomially distributed errors. It does not matter whether the output file 'error.seq' exists or not. *If it exists, it is overwritten without a prompt to the user.*

There are no assumptions associated with the implementation or output of this program.

2. BrstErrS (Burst Error Sequence generation)

This program generates a binary error sequence with burst errors. The error sequence denotes a correct binary channel transmission with a 0 and denotes an error with a 1. The error sequence is partitioned into two main, noncontiguous parts, the burst error part and the error free part. The method used to generate the burst error part of the error sequence depends upon the density of errors to be generated. If the required density of errors is greater than 0.01, then the program uses a conditional test on a uniform random number in the range [0,1]. If the density of errors is less than 0.01, then the program uses a sample from the exponential distribution to generate the next error occurrence time.

This program inputs parameters from an ASCII data file with default name 'BurstErrs.prm' and outputs the error sequence to a data file with default name 'error.seq'. In addition, various statistics are output to an ASCII data file with default name 'BurstErrs.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'BurstErrs.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program generates the 'error.seq' file which contains an error sequence (in packed format) with binomially distributed errors. It does not matter whether the output file 'error.seq' exists or not. *If it exists, it is overwritten without a prompt to the user.*

Even though Poisson distributed bursts may overlap in theory, this program does not allow error bursts to overlap. The user must take care to specify input parameters so that the probability of overlapping bursts is negligible.

3. BstyErrs (Bursty Errors Sequence generation)

This program generates an binary error sequence with bursty errors; that is, a combination of random and burst errors. The error sequence denotes a correct binary channel transmission with a 0 and denotes an error with a 1. The error sequence is partitioned into two main, noncontiguous parts, the burst error part and the random error part. The method used to generate each part of the error sequence depends upon the density of errors to be generated. For each error sequence part, if the required density of errors is greater than 0.01, then the program uses a conditional test on a uniform random number in the range [0,1]. If the density of errors is less than 0.01, then the program uses a sample from the exponential distribution to generate the next error occurrence time.

This program inputs parameters from an ASCII data file with default name 'BurstyErrs.prm' and outputs the error sequence to a data file with default name 'error.seq'. In addition, various statistics are output to an ASCII data file with default name 'BurstyErrs.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'BurstyErrs.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program generates the 'error.seq' file which contains an error sequence (in packed format) with binomially distributed errors. It does not matter whether the output file 'error.seq' exists or not. *If it exists, it is overwritten without a prompt to the user.*

Even though Poisson distributed bursts may overlap in theory, this program does not allow error bursts to overlap. The user must take care to specify input parameters so that the probability of overlapping bursts is negligible. It is also assumed that $P_{eg} < P_{eb}$.

C. Interleavers

The contract requires that block interleavers and periodic convolutional interleavers be considered. To this end, a program is described which implements the effect of a block interleaver and a separate program is described which implements the effect of a periodic convolutional interleaver. Also, there are two versions of each program. The two versions implement the same operation but trade off computer code complexity for execution speed.

1. BlockInt (Block Deinterleaver)

This program performs block deinterleaving of the error sequence found in file 'error.seq'. It is assumed that the channel symbols corresponding to those errors have already been interleaved using an (C,R,m) block interleaver. The deinterleaver groups every m error sequence values together and deinterleaves them as a group. The method used to implement the function of the block interleaver is to read in a block of the error sequence and to use a series of formulas to perform the block deinterleaving. These formulas are described below.

Let b_K denote the error sequence input to the deinterleaver and let d_L denote the error sequence output by the deinterleaver. Note: the subscripts are assumed to be incremented starting with zero. Then b_K is read into the deinterleaver memory array (by columns) at location:

Symbol index = int(K/m) == Y
Row of b_K = Mod(Y,R) == i
Column of b_K = int(Y/R) == j
Depth of b_K = Mod(K,m) == p

Given i, j, and p the deinterleaved value location (read out by rows) is found to be

$$L = m * (i*C+j) + p$$

The implementation found below actually calculates K given L. The actual value b_K is found in a buffer which is loaded with error sequence values. The calculation is as follows:

1) L points to location BuffL in the buffer, BuffL = Mod(L,BuffLength)

2) The interleaved location for BuffL is BuffK where

ll = Mod(BuffL,m)
X = BuffL/m
BuffK = m * (R*Mod(X,C) + int4(X/C)) + ll

where BuffLength=$R*C*m$. Note that there is a problem deinterleaving the end of the 'error.seq' file due to a possible partial interleaver block at the end of the sequence. The program attempts to partially deinterleave this last partial block. An error sequence could be zero padded to fill a partial block, thereby changing slightly the overall error statistics.

This program inputs parameters from an ASCII data file with default name 'BlockInt.prm' and outputs the error sequence to data file with default name 'error.seq'. In addition, various statistics are output to an ASCII data file with default name 'BlockInt.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'BlockInt.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program generates the 'error.seq' file which contains an error sequence (in packed format) with deinterleaved errors. *The 'error.seq' file must exist prior to the execution of this program.*

There are no assumptions associated with the implementation or output of this program.

2. BlkArr (Alternate Block Deinterleaver)

This program performs block deinterleaving of the error sequence found in file 'error.seq'. It is assumed that the channel symbols corresponding to those errors have already been interleaved using an (C,R,m) block interleaver. The deinterleaver groups every m error sequence values together and deinterleaves them as a group. The method used to implement the function of the block interleaver is to read in a block of the error sequence into a buffer which mimics the block interleaver memory array. The error sequence is read in by rows and deinterleaving is performed by reading the error sequence out by columns.

This program inputs parameters from an ASCII data file with default name 'BlockInt.prm' and outputs the error sequence to data file with default name 'error.seq'. In addition, various statistics are output to an ASCII data file with default name 'BlockInt.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'BlockInt.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program generates the 'error.seq' file which contains an error sequence (in packed format) with deinterleaved errors. *The 'error.seq' file must exist prior to the execution of this program.*

There are no assumptions associated with the implementation or output of this program.

3. DPCI (Periodic Convolutional Deinterleaver)

This program performs deinterleaving of the error sequence found in file 'error.seq'. It is assumed that the channel symbols corresponding to those errors have already been interleaved using an (Ntaps,M) periodic convolution interleaver. The method used to implement the function of the periodic convolutional interleaver is a series of formulas as described below. These functions are applied to a portion of the error.seq array which is stored in a ring buffer.

Let b_K denote the error sequence input to the DPCI and let d_L denote the error sequence output by the DPCI. Then the index L relates to the index K as follows,

K = Mod((L-1),Ntaps) * M * Ntaps + L

Note that there is a problem deinterleaving the end of the 'error.seq' file due to the sequential nature of the algorithm. The DPCI error sequence file is truncated to eliminate the "don't cares".

This program inputs parameters from an ASCII data file with default name 'DPCI.prm' and outputs the error sequence to data file with default name 'error.seq'. In addition, various statistics are output to an ASCII data file with default name 'DPCI.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'DPCI.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program generates the 'error.seq' file which contains an error sequence (in packed format) with deinterleaved errors. *The 'error.seq' file must exist prior to the execution of this program.*

There are no assumptions associated with the implementation or output of this program.

4. DPCIAlt (Alternate Periodic Convolutional Deinterleaver)

This program performs deinterleaving of the error sequence found in file 'error.seq'. It is assumed that the channel symbols corresponding to those errors have already been interleaved using an (n,M) periodic convolution interleaver. The method used to implement the function of the periodic convolutional interleaver is a series of formulas as described below.

Let b_i denote the error sequence input to the DPCI and let d_j denote the error sequence output by the DPCI. Then the index j relates to the index i as follows,

$j = i - [(i-1) \bmod n]*M*n$

Note that there is a problem deinterleaving the end of the 'error.seq' file due to the sequential nature of the algorithm. For this case, the 'error.seq' file is filled with zeroes for those deinterleaved positions which result from locations which are beyond the end of the 'error.seq' file.

This program inputs parameters from an ASCII data file with default name 'DPCI.prm' and outputs the error sequence to data file with default name 'error.seq'. In addition, various statistics are output to an ASCII data file with default name 'DPCI.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'DPCI.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program generates the 'error.seq' file which contains an error sequence (in packed format) with deinterleaved errors. *The 'error.seq' file must exist prior to the execution of this program.*

There are no assumptions associated with the implementation or output of this program.

D. Error Sequence Analysis

The contract requires that error sequence be characterized. This amounts to modeling the errors by a predefined mathematical model. Several mathematical models are considered; one which models the errors as bursty errors, and one which models the errors as burst errors. Bursty errors are characterized by errors which occur within bursts as well as errors which occur outside bursts. Burst errors are characterized by errors which occur only within bursts. In addition, two programs have been written to implement distribution tests for the purpose of determining if an error sequence, or a segment of an error sequence, resulted from random errors.

1. CVMseq (Cramer Von-Mises sequence distribution test)

This program uses the Cramer Von-Mises (CVM) distribution test to determine whether the error sequence (in default file 'error.seq') is binomially distributed with confidence level alpha. The method implemented is simple. The error sequence is read in by blocks and the overall CVM test statistic is calculated. At the end of the program, the test statistic for the complete sequence along with a preselected set of critical values is output to the user. The results are also output to 'CVMseq.ID' file where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

Executing the program causes the 'error.seq' file to be read which contains an error sequence (in packed format). *The 'error.seq' file must exist prior to the execution of this program.* There are no assumptions associated with the implementation or output of this program.

2. CVMblk (Cramer Von-Mises distribution test on error sequence blocks)

This program uses the Cramer Von-Mises (CVM) distribution test to determine whether the error sequence (in default file 'error.seq') is binomially distributed with confidence level alpha. The error sequence is read in by blocks and the CVM test statistic is calculated for each block. At the end of the program, the test statistics for each block along with a preselected set of critical values are ordered and output to the user. The results are also output to 'CVMblk.ID' file where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

Executing the program causes the 'error.seq' file to be read which contains an error sequence (in packed format). *The 'error.seq' file must exist prior to the execution of this program.* There are no assumptions associated with the implementation or output of this program.

3. DeltaEst (Bursty-error parameter estimation via the Δ method)

This program estimates parameters associated with a bursty error sequence. The method employed segments the error sequence into random error regions and burst error regions. The algorithm implemented operates on the error sequence iteratively. For each iteration, the algorithm is either tracking a burst segment or a random segment. At each iteration, the error sequence interval to the next error is found. If the algorithm is tracking a random segment, then an attempt is made to begin a burst by comparing the error density for the i^{th} interval (surrounded by 2 errors which gives an effective error density of 2/[interval+2]) with a threshold (Delta). If the error density for the i^{th} interval is greater than Delta, then the algorithm begins tracking a burst segment, if not then the random segment is continued. If the algorithm is tracking a burst segment, then the segment is continued until the error density within the total burst segment falls below the threshold, Delta. In this way, the entire sequence is partitioned. Initializing the processes is particularly troublesome because of the various combinations for the beginning of the error seq.

This program inputs parameters from an ASCII data file with default name 'DeltaEst.prm' and outputs the error sequence to data file with default name 'error.seq'. In addition, various statistics are output to an ASCII data file with default name 'DeltaEst.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'DeltaEst.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program generates the 'error.seq' file which contains an error sequence (in packed format) with deinterleaved errors. *The 'error.seq' file must exist prior to the execution of this program.*

There are no assumptions associated with the implementation or output of this program.

4. IntvHst (Error Interval Histogram)

This program calculates the error interval probability density function for an error sequence. The error sequence is partitioned into error free segments and a histogram of the interval length calculated. Note that the two error free intervals occurring at the beginning of the error sequence and at the end are ignored. Only intervals between errors are counted.

The program outputs the histogram to file 'Interval.hst' which (for now) is an ASCII file with each histogram value stored per record. For each record, the interval index appears first followed by the probability of occurrence.

Note that there are NO parameters to be read in for this program. However, various statistics are output to an ASCII data file with default name 'IntvHst.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'IntvHst.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program generates the 'Interval.hst' file which contains the histogram of the error intervals found in the error sequence. *The 'error.seq' file must exist prior to the execution of this program.*

There are no assumptions associated with the implementation or output of this program.

5. GAPEst (fixed GAP burst error distribution Estimation)

This program estimates parameters associated with a bursty error sequence. The method employed segments the error sequence into error free regions and burst error regions. A burst error region is defined to be a region which contains errors no two of which are separated by more than the prespecified GAP number of error free symbols. In addition, the burst error region is preceded and followed by error free regions of minimum width specified by GAP. The algorithm implemented operates on the error sequence iteratively. For each iteration, the algorithm determines the width of the next error free interval, if it is less than GAP then the next error is included in the current burst, if it is greater than GAP then the previous burst is terminated and the next burst is started. In this way, the entire sequence is partitioned. If the first error sequence value is a '0' then the process always begins with an error free region. If the first error sequence value is a '1' then the process always begins with an error burst.

This program inputs parameters from an ASCII data file with default name 'GAPEst.prm' and outputs the error sequence to data file with default name 'error.seq'. In addition, various statistics are output to an ASCII data file with default name 'GAPEst.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'GAPEst.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. Executing the program causes the 'error.seq' file to be read which contains an error sequence (in packed format). *The 'error.seq' file must exist prior to the execution of this program.*

There are no assumptions associated with the implementation or output of this program.

E. Utilities

Several utilities have been developed to support CLEAN. The *makefile* given in the following section can be used to compile the source code with a single command by typing 'make all'. The programs which follow allow the user to compare error sequences, set error sequences, and display error sequences.

1. Make Utility for Lahey Fortran v5.0

```
FFLAGS = /3 /B /nA1 /C1 /P /R /Z1

CorrCW   = CorrCW.obj Unpack.obj Pack.obj
GaussRV  = GaussRV.obj UnifRV.obj
IterBin  = IterBin.obj UnifRV.obj
LdBuff1  = LdBuff1.obj Unpack.obj
LdBuff4  = LdBuff4.obj Unpack.obj
NextBrst = NextBrst.obj UnifRV.obj GaussRV.obj
NextInt  = NextInt.obj LdBuff4.obj
NextLnth = NextLnth.obj UnifRV.obj GaussRV.obj
SvBuff1  = SvBuff1.obj Unpack.obj Pack.obj
SvBuff4  = SvBuff4.obj Unpack.obj Pack.obj
TotalPe  = TotalPe.obj Unpack.obj

ALL : BinErrs BlkArr BlkDecod Blockint BrstErrs \
      BstyErrs CompSeq CVMblk CVMseq DeltaEst DisplFil DisplSeq \
      DPCI DPCIOld GAPest IntvHst SetErrs

BINERRS : BINERRS.obj $(IterBin) Pack.obj UnifRV.obj
  Optlink BINERRS.obj $(IterBin) Pack.obj UnifRV.obj, \
          BINERRS.exe,,c:\compiler\lahey\F77L.LIB

BLKARR :  BLKARR.obj $(LdBuff1) $(SvBuff1) DispBuf1.obj
  Optlink BLKARR.obj $(LdBuff1) $(SvBuff1) DispBuf1.obj, \
          BLKARR.exe,,c:\compiler\lahey\F77L.LIB

BLKDECOD : BLKDECOD.obj $(LdBuff4) $(CorrCW)
  Optlink  BLKDECOD.obj $(LdBuff4) $(CorrCW), \
           BLKDECOD.exe,,c:\compiler\lahey\F77L.LIB

BLOCKINT : BLOCKINT.obj $(LdBuff1) $(SvBuff1) Pack.obj
  Optlink  BLOCKINT.obj $(LdBuff1) $(SvBuff1) Pack.obj, \
           BLOCKINT.exe,,c:\compiler\lahey\F77L.LIB

BRSTERRS : BRSTERRS.obj $(IterBin) Pack.obj UnifRV.obj $(NextBrst) $(NextLnth)
  Optlink  BRSTERRS.obj $(IterBin) Pack.obj UnifRV.obj $(NextBrst) $(NextLnth) , \
           BRSTERRS.exe,,c:\compiler\lahey\F77L.LIB

BSTYERRS : BSTYERRS.obj $(IterBin) Pack.obj UnifRV.obj $(NextBrst) $(NextLnth)
  Optlink  BSTYERRS.obj $(IterBin) Pack.obj UnifRV.obj $(NextBrst) $(NextLnth) , \
           BSTYERRS.exe,,c:\compiler\lahey\F77L.LIB
```

```
COMPSEQ: COMPSEQ.obj Unpack.obj
  Optlink COMPSEQ.obj Unpack.obj, \
          COMPSEQ.exe,,c:\compiler\lahey\F77L.LIB
CVMblk :  CVMblk.obj $(LdBuff4) RdStats.obj
  Optlink CVMblk.obj $(LdBuff4) RdStats.obj, \
          CVMblk.exe,,c:\compiler\lahey\F77L.LIB
CVMseq:  CVMseq.obj $(LdBuff4) $(NextInt) RdStats.obj
  Optlink CVMseq.obj $(LdBuff4) $(NextInt) RdStats.obj, \
          CVMseq.exe,,c:\compiler\lahey\F77L.LIB
DELTAEST : DELTAEST.obj $(LDBuff4) $(NextInt) $(TotalPe)
  Optlink  DELTAEST.obj $(LDBuff4) $(NextInt) $(TotalPe) , \
           DELTAEST.exe,,c:\compiler\lahey\F77L.LIB
DISPLFIL : DISPLFIL.obj Unpack.obj
  Optlink  DISPLFIL.obj Unpack.obj, \
           DISPLFIL.exe,,c:\compiler\lahey\F77L.LIB
DISPLSEQ: DISPLSEQ.obj Unpack.obj
  Optlink  DISPLSEQ.obj Unpack.obj, \
           DISPLSEQ.exe,,c:\compiler\lahey\F77L.LIB
DPCI :    DPCI.obj Unpack.obj Pack.obj
  Optlink DPCI.obj Unpack.obj Pack.obj, \
          DPCI.exe,,c:\compiler\lahey\F77L.LIB
DPCIOLD : DPCIOLD.obj Unpack.obj Pack.obj
  Optlink DPCIOLD.obj Unpack.obj Pack.obj, \
          DPCIOLD.exe,,c:\compiler\lahey\F77L.LIB
GAPEST : GAPEST.obj $(LdBuff4) $(NextInt)
  Optlink  GAPEST.obj $(LdBuff4) $(NextInt) , \
           GAPEST.exe,,c:\compiler\lahey\F77L.LIB
IntvHst :  IntvHst.obj $(LDBuff4) $(NextInt)
  Optlink  IntvHst.obj $(LDBuff4) $(NextInt) , \
           IntvHst.exe,,c:\compiler\lahey\F77L.LIB
SETERRS : SETERRS.obj Pack.obj
  Optlink SETERRS.obj Pack.obj, \
          SETERRS.exe,,c:\compiler\lahey\F77L.LIB


BINERRS.obj : BINERRS.for
  F77L BINERRS.for $(FFLAGS)
BLKARR.obj : BLKARR.for
  F77L BLKARR.for $(FFLAGS)
BLKDECOD.obj : BLKDECOD.for
  F77L BLKDECOD.for $(FFLAGS)
BLOCKINT.obj : BLOCKINT.for
  F77L BLOCKINT.for $(FFLAGS)
BRSTERRS.obj : BRSTERRS.for
  F77L BRSTERRS.for $(FFLAGS)
BSTYERRS.obj : BSTYERRS.for
  F77L BSTYERRS.for $(FFLAGS)
COMPSEQ.obj : COMPSEQ.for
  F77L COMPSEQ.for $(FFLAGS)
CVMblk.obj : CVMblk.for
  F77L CVMblk.for $(FFLAGS)
CVMseq.obj : CVMseq.for
  F77L CVMseq.for $(FFLAGS)
CorrCW.obj : CorrCW.for
  F77L CorrCW.for $(FFLAGS)
DELTAEST.obj : DELTAEST.for
  F77L DELTAEST.for $(FFLAGS)
DispBuf1.obj : DispBuf1.for
  F77L DispBuf1.for $(FFLAGS)
DispBuf4.obj : DispBuf4.for
  F77L DispBuf4.for $(FFLAGS)
DISPLFIL.obj : DISPLFIL.for
  F77L DISPLFIL.for $(FFLAGS)
DISPLSEQ.obj : DISPLSEQ.for
  F77L DISPLSEQ.for $(FFLAGS)
DPCI.obj : DPCI.for
  F77L DPCI.for $(FFLAGS)
DPCIOLD.obj : DPCIOLD.for
  F77L DPCIOLD.for $(FFLAGS)
```

```
GAPEST.obj : GAPEST.for
  F77L GAPEST.for $(FFLAGS)
GAUSSRV.obj : GAUSSRV.for
  F77L GAUSSRV.for $(FFLAGS)
IntvHst.obj : IntvHst.for
  F77L IntvHst.for $(FFLAGS)
ITERBIN.obj : ITERBIN.for
  F77L ITERBIN.for $(FFLAGS)
LDBUFF1.obj : LDBUFF1.for
  F77L LDBUFF1.for $(FFLAGS)
LDBUFF4.obj : LDBUFF4.for
  F77L LDBUFF4.for $(FFLAGS)
NEXTBRST.obj : NEXTBRST.for
  F77L NEXTBRST.for $(FFLAGS)
NEXTINT.obj : NEXTINT.for
  F77L NEXTINT.for $(FFLAGS)
NEXTLNTH.obj : NEXTLNTH.for
  F77L NEXTLNTH.for $(FFLAGS)
PACK.obj : PACK.for
  F77L PACK.for $(FFLAGS)
RdStats.obj : RdStats.for
  F77L RdStats.for $(FFLAGS)
SETERRS.obj : SETERRS.for
  F77L SETERRS.for $(FFLAGS)
SVBUFF1.obj : SVBUFF1.for
  F77L SVBUFF1.for $(FFLAGS)
SVBUFF4.obj : SVBUFF4.for
  F77L SVBUFF4.for $(FFLAGS)
TotalPe.obj : TotalPe.for
  F77L TotalPe.for $(FFLAGS)
UNIFRV.obj : UNIFRV.for
  F77L UNIFRV.for $(FFLAGS)
UNPACK.obj : UNPACK.for
  F77L UNPACK.for $(FFLAGS)
```

2. CompSeq (Compare Sequence)

This program compares two error sequences and identifies those error locations where the two are different. The user is prompted for the two error sequence filenames. It is assumed that the errors stored in error.seq are in the DBESS (Double Byte Error Sequence Symbol) packed format.

3. SetErrS (Set Error Pattern)

This program interactively allows the user to input an error sequence. All parameters and the error sequence are input directly from the user so that there is no parameter file associated with this program. The errors are stored in the DBESS packed format.

There are no assumptions associated with the implementation or output of this program.

4. DisplSeq (Display Default Error Sequence)

This program displays the error sequence found in file 'error.seq'. It is assumed that the errors stored in error.seq are in the DBESS packed format.

5. DisplFil (Display Error Sequence from user File)

This program displays the error sequence found in a file specified by the user. It is assumed that the errors stored in the file are in the DBESS packed format.

IV. NASA GSFC/MSU INTERRELATED CAPABILITIES

To enhance the research efforts at both MSU and NASA GSFC, several interrelated capabilities have been established. The first author visited GSFC in August of 1992 to learn how to use the Communications Link And System Simulation (CLASS) software tool. CLASS performs a signal level simulation of the TDRS downlink and predicts coded system performance using theoretical analysis. In addition, the first author learned how to use the OMV bit-by-bit simulator which uses the same signal level simulation nucleus as CLASS but also incorporates actual deinterleaving and decoding algorithms to simulate the operation of the deinterleavers and decoders at White Sands. After learning how to use these software tools, analyst level access was granted and has been established. It is now possible for MSU personnel to exercise CLASS and the OMV bit-by-bit simulator remotely from MSU via internet. MSU appreciates the support given by the NASA/GSFC CLASS group.

Furthermore, real EOS Ku-band downlink data (validity of the data pending) has been acquired by Victor Sank at GSFC. A program was written to convert from the GSFC error sequence data format into the format required by CLEAN. Since these data files are sometimes rather large which requires large storage spaces, a second program was written to archive the GSFC data using run length encoding, a *lossless* compression scheme. For an error sequence with an error probability of 10^{-3}, this provides about 3:1 lossless compression. For an error sequence with an error probability of 10^{-4}, this provides about 30:1 lossless compression. In addition, a third program was written to unarchive the run length encoded data into the DBESS format required by CLEAN. Mr. Sank's help has been invaluable to this project.

A. EOS Real Error Sequence Data conversion program

This program inputs the real EOS downlink data obtained from Victor Sank and converts it into the DBESS packed format required by the programs in CLEAN.

It is assumed that the input file accessed by this program exists prior to its execution.

B. Error Sequence Archiver using Run Length Encoding

This program inputs the real EOS downlink data obtained from Victor Sank and converts it into an archival format. The archival format only stores the location of each error in the file. This is *not* the format which is necessary for CLEAN. Another program called SeqUnarc can be executed to convert from the Archival format to the DBESS format required by CLEAN.

It is assumed that the input file accessed by this program exists prior to its execution.

C. Error Sequence Unarchiver

This program inputs data in the archival format (run length encoding) via the SeqArc program and unarchives it to the DBESS format required by CLEAN.

It is assumed that the input file accessed by this program exists prior to its execution.

V. PREVIEW OF EXPECTED RESULTS

The problem of interest is that of choosing/evaluating a good forward error correcting coding (FEC) scheme for the Ku-band TDRS downlink which will be used for the Earth Observation System (EOS). There are many issues to be considered when choosing a "good" FEC including required error probability, required data rate, and data loss during synchronization cycles just to name a few.

For example, suppose it is proposed to use a (255,223) Reed-Solomon (RS) code with a block interleaver for the 150Mbps Ku-band TDRS downlink. If this code meets the required error probability, say 10^{-5}, for the types and density of errors anticipated on the link and if it can accommodate the required data rate, 150Mbpsx(223/255)=131Mbps, then this code can be considered acceptable. If the decision is made to concatenate a rate 1/2 convolutional encoder and periodic convolutional interleaver with the RS code and block interleaver, then several undesirable side effects will take place. First, the hardware complexity will increase which will increase cost, size, weight, power, etc. Second, the periodic convolutional deinterleaver and Viterbi decoder at the receiver must synchronize to the received data. The synchronization process can result in significant data loss. In addition, the convolutional code rate results in a decrease in the system data rate to 131Mbpsx(1/2)=65.6Mbps, assuming a fixed channel rate. Although this concatenated scheme may provide a lower error probability which exceeds the requirement, it is achieved at a significant cost. Therefore, the studies developed for this contract focus on determining and evaluating the minimum complexity coding scheme for EOS to satisfy the system requirements. This requires an understanding of the nature of the Ku-band downlink errors and of the achievable performance for various coding schemes in various types of error environments.

To this end, the research is being focussed along two main lines as discussed in the following sections.

A. Research Focus 1

First, the nature of the downlink errors is being investigated. The expected results are a consequence of discussions with NASA/GSFC and STEL personnel concerning the nature of the Ku-band downlink errors. The expected results are:

1) Determine that the expected errors which occur in a received block of data are *not* random. This is accomplished by applying the Cramer Von-Mises distribution test (see CVMblk in Section III.D.2) to the actual data.

2) Estimate the error parameters for the actual channel data assuming that the errors are bursty in nature. These will be estimated by applying the bursty-error parameter estimation via the Δ method (see DeltaEst in Section III.D.3) to the actual data. It is expected that the burst locations follow a Poisson distribution. The estimated parameters are:

   a) Average rate of burst occurrence and the burst occurrence interval probability density function (*pdf*). It is expected that this *pdf* is exponential which means that the burst locations follow a Poisson distribution.
   b) The average burst length (in channel symbols) and the burst length *pdf.* It is expected the variance of this *pdf* is small.
   c) The average error density during the bursts and the burst error density *pdf.* It is expected that the variance of this *pdf* is small.
   d) The average error density outside the bursts. It is expected that this error density will be very nearly the random error rate.

Because the actual data has not been received to date, this work has not been completed.

B. Research Focus 2

The second focus of this research is the investigation of performance for various coding schemes in a bursty-error environment. The expected result will be plots similar to the one shown in Figure 2. Several coding schemes will be considered including:

1) Reed-Solomon (RS)
2) RS, block interleaver (interleave depth of 5)
3) RS, block interleaver (interleave depth of 8)
4) RS outer code, block interleaver (interleave depth of 5), convolutional inner code
5) RS outer code, block interleaver (interleave depth of 5), convolutional inner code, periodic convolutional interleaver.

The curves drawn are for illustration only but *do* indicate to some degree the expected shape. The *error ratio* R_ε, as defined in this research, is

$$R_\varepsilon = \frac{\text{Total Random Errors}}{\text{Total Errors}}$$

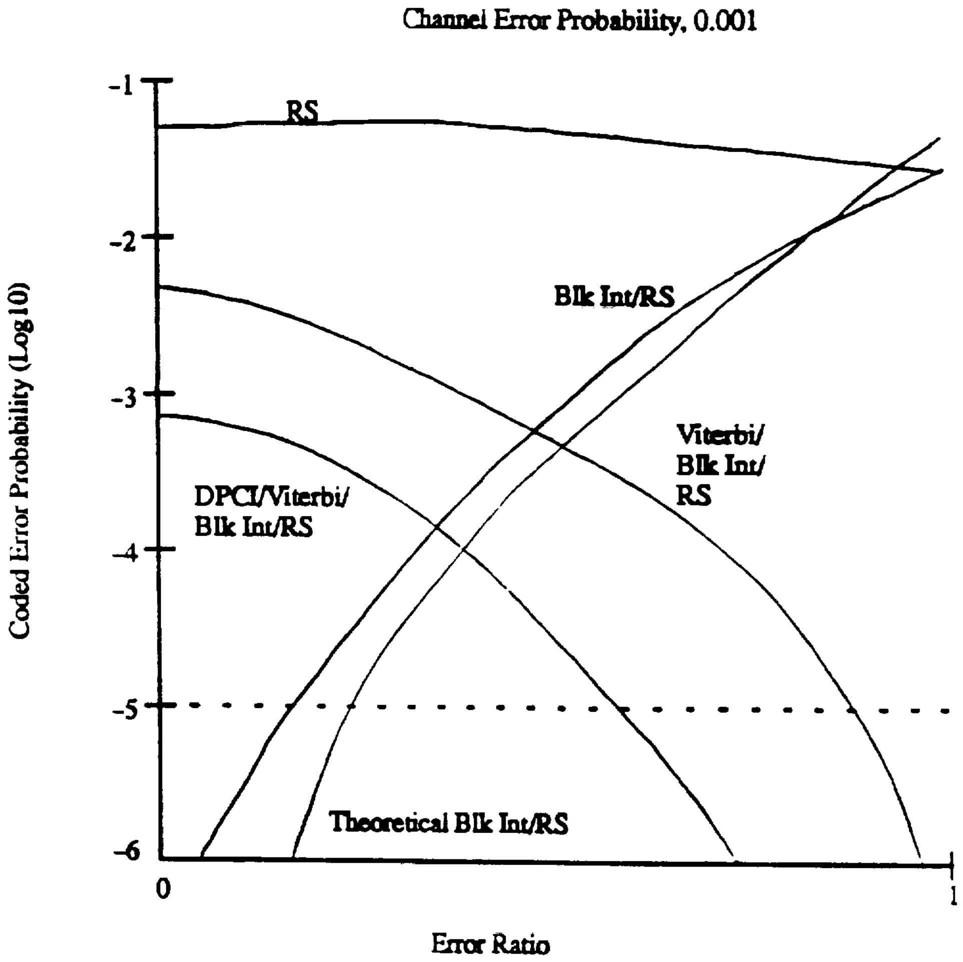


Figure 2. An expected output performance data product (for illustration only).

To construct Figure 2, a channel error probability, $P(\varepsilon_{ch})$, is chosen. For each possible error rate, the bursty-error parameters are calculated and CLEAN is used to calculate the decoded error probability. For example, to simulate system (5) identified above, the following programs are sequentially executed:

1) BstyErrs (see Section III.B.3)
2) DPCI (see Section III.C.3)
3) Viterbi (see Section III.A.2)
4) BlockInt (see Section III.C.1)
5) BlkDecod (see Section III.A.1)

The input parameters must be chosen and input to the appropriate parameter files. The choice for the input parameters are discussed in the following section. The file 'BlkDecod.ID' where ID is the 3 letter identifier found in file 'ID.prm' gives the final decoded error probability. Note that CLEAN performs a Monte Carlo simulation.

It is expected that the actual plot, similar to that shown in Figure 2, will show that the Reed-Solomon code used with a block interleaver (interleave depth of 5) is sufficient to provide the required decoded error probability and, therefore, constitutes the "best" coding scheme.

To date, about 10% of the actual plot has been developed for the choice of parameters discussed in the following section. The required execution time of some of the programs is on the order of hours per data point for a SPARC workstation.

C. Choosing System Parameters

Of interest in this research are performance results for codes which are used for space based communication systems. The Consultative Committee for Space Data Systems (CCSDS) [1] defines a concatenated coding scheme for space based communication systems consisting of a (255,223) RS outer code followed by an interleaver and a rate 1/2 constraint length 7 convolutional inner code. Therefore, these are the code parameters chosen for study in this research. To summarize

1) Reed-Solomon code (BlkDecod program)
    a) Blocklength, n=255
    b) Information codeword length, k=223
    c) Number of binary symbols per codeword, m=8
    c) Error correcting capability, t=16 code symbols per codeword
2) Convolutional code (Viterbi program)

a) Constraint length, K=7

b) Number of code generators, 2 (code rate = 1/2)

c) Tap weights for code generator #1, 1011011

d) Tap weights for code generator #2, 1111001

e) Number of constraint lengths for decoder memory, 4

Also of interest are the interleaver parameters. The Framing and Multiplex Equipment (FAME) defines a standard architecture for space based communication systems which involves multiplexing 8 (only 5 are utilized) data streams together to form a single data stream for transmission to earth. This results in a block interleaving effect for the demultiplexed data input to the RS decoder. Therefore, the block interleaver imitates the multiplex operation. For the (255,223) RS code defined above, this requires the block interleaver parameters to be chosen as

3) Block interleaver (BlockInt)

a) Number of rows, 5 (This is alternately chosen to be 8)

b) Number of columns, 255

c) Number of binary symbols per memory array element, 8

In addition, the periodic convolutional interleaver currently used has parameters given by

4) Periodic Convolutional Interleaver (DPCI)

a) Number of taps, 30

b) Number of delays for the 2nd tap, 2

The only parameters remaining to be specified are the bursty-error parameters. This requires choosing the burst duration *pdf* be chosen along with the mean and possibly the variance, the burst location *pdf* be chosen along with the mean and possibly the variance, the error density within the bursts, and the error density outside the bursts. These parameters must be chosen for the given raw channel error probability, $P(\varepsilon_{ch})$, and for each possible value for the *error ratio*.

It is known that the Ku-band downlink is characterized by essentially error free transmission interrupted by short, fixed periods of high interference. The interference is probably less than 0.3μsec in duration. Although the average time between error bursts is unknown, the duty cycle of the interference is probably less than 0.025. Given this information, a worse case scenario can be constructed. If the worse case interference duration is 0.3μsec and the channel symbol rate is 75Mbps (2 binary symbols per channel symbol for QPSK gives rise to the required 150Mbps), then $(0.3\text{x}10^{-6})(75\text{x}10^{6}\text{bps})$(2bits/channel symbol)=45 binary symbols is the length of each error burst. As an aside, it is easy to determine that a (255,223) RS code with a depth 5 block interleaver can correct an error burst of 45 binary symbols. However, it is possible for multiple error bursts to occur within one interleaved block. In light of this characterization, some of the bursty-error parameters are chosen as follows

5) Bursty-error Generation (BstyErrS)
   a) Burst occurrence location *pdf*, IntvFlag=3 (Poisson)
   b) Burst occurrence interval mean, IntvMean=4500 binary symbols
   c) Burst occurrence duration *pdf*, LngthFlag=1 (Fixed)
   d) Burst occurrence duration mean, LngthMean=45 binary symbols

The only two parameters remaining to be chosen are the error probability during the error bursts, P_{eb}, and the error probability outside the error bursts, P_{eg}. Choosing these is more involved than the previous parameters because they must be calculated for the predefined raw channel error probability, $P(\varepsilon_{ch})$, and because they must be changed to adjust the *error ratio*.

The method for calculating P_{eb} and P_{eg} in terms of $P(\varepsilon_{ch})$ and R_ε is as follows. From [2], the raw channel error probability for a bursty-error channel is given by

$$P(\varepsilon_{ch}) = P_{eg}(1 - d/M_v) + P_{eb}(d/M_v)$$

where M_v is the average interval between error bursts (denoted IntvMean in part 4.b above) and where d is the burst duration (denoted LngthMean in part 4.d above). The error ratio can be expressed in terms of these symbols to be

$$R_\varepsilon = \frac{P_{eg}(1 - d/M_v)}{P(\varepsilon_{ch})}$$

Solving the previous two equations for P_{eg} and P_{eb} gives

$$P_{eg} = \frac{R_\varepsilon P(\varepsilon_{ch})}{1 - d/M_v}$$

and

$$P_{eb} = \frac{M_v}{d}(1 - R_\varepsilon)P(\varepsilon_{ch})$$

These are valid provided $P_{eb} \geq P_{eg}$. Note that for given values of d and M_v, it is generally not possible for the *error ratio* to take on all values from 0 to 1. Clearly, $P_{eg} \leq P_{eb} \leq 1/2$, from which it can be determined that

$$0 \leq \frac{P(\varepsilon_{ch}) - d/(2M_v)}{P(\varepsilon_{ch})} \leq R_\varepsilon \leq 1 - d/M_v \leq 1$$

which implies that we must have

$$P(\varepsilon_{ch}) \geq \frac{d}{2M_v}$$

Note that if we choose $P(\varepsilon_{ch}) = d/(2M_v)$ then it is possible to achieve a range for the error ratio of $0 \leq R_\varepsilon \leq 1$ by selecting appropriate values for P_{eg} and P_{eb}.

BIBLIOGRAPHY

1. Consultative Committee for Space Data System, "Recommendations for space data system standards: Telemetry channel coding." *Blue Book*, May 1984.

2. Ebel, W.J., Simulation and Evaluation of Reed-Solomon Codes in a Burst Noise Environment, Ph.D. Dissertation, University of Missouri-Rolla, 1991.

APPENDIX II

An Investigation of Error Characteristics and Coding Performance

NASA GRANT NAG5-2006

Annual Report

January 1993 - August 1993

An Investigation of
Error Characteristics and Coding Performance

NASA GRANT NAG5-2006
July 1, 1992 - June 30, 1993

Quarterly Report
January 1, 1993 - March 30, 1993

Submitted to:

Mr. Warner Miller
Code 728.4
Instrument Electronic Systems Branch
Engineering Directorate
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
301-286-8183

Submitted by:

William J. Ebel, Ph.D.
Frank M. Ingels, Ph.D.
Mississippi State University
Drawer EE
Mississippi State, MS 39762
601-325-3912

March 1993

Mississippi State
UNIVERSITY

Department of Electrical and Computer Engineering
P.O. Drawer EE
Mississippi State, MS 39762-5660
Tel: (601) 325-3912 FAX: (601) 325-2298

April 8, 1993

Mr. Warner Miller
MS Code 728
NASA Goddard Space Flight Center
Greenbelt, MD 20771

Dear Warner:

Enclosed you will find three copies of the quarterly report for the period January 1, 1993 to March 30, 1993 for NASA Grant NAG5-2006.

In my last letter that we FAXed to you dated March 12, we mentioned that we saw an anomaly in the data we received from GSFC (Victor Sank). It turns out that the anomaly was simply a bug in the Lehay FORTRAN compiler that we are using which caused a format conversion error. However, the format conversion error only caused a very small fraction of spurious errors to be imbedded in the error sequence. In fact, the fraction of spurious errors was so small that it did not have a significant influence on the computed statistics. Although the data used for this report is for the slightly modified data, the conclusions drawn are true to the actual data generated by GSFC.

After identifying the bug and fixing the problem, no data inversions or other anomalies in the data were found. All results for the final report will be based upon the correct error sequence data.

We look forward to your comments.

Sincerely yours,

William J. Ebel

Will Ebel
Assistant Professor

Frank Ingels

Frank Ingels
Professor
Electrical and Computer Engineering

Mississippi State
UNIVERSITY

Department of Electrical and Computer Engineering
P.O. Drawer EE
Mississippi State, MS 39762-5660
Tel: (601) 325-3912 FAX: (601) 325-2298

April 8, 1993

NASA Science & Technology Information Facility
ATTN: Accessioning Department
800 Elkridge Landing Road
Linthicum Heights, MD 21090

TO WHOM IT MAY CONCERN:

Enclosed are two (2) copies of the quarterly report for the NASA Grant NAG5-2006.

Sincerely,

Will Ebel

Will Ebel
Assistant Professor

Frank Ingels

Frank Ingels
Professor
Electrical and Computer Engineering

Abstract

In February of 1993, real error sequence data from the Ku-band downlink was obtained from NASA/GSFC. The data consists of the error sequence found at the NRZM decoder output. Along with theoretical results, some processing of this data has been completed and is reported in this document.

In this report, performance evaluation of various coding schemes operating on bursty errors is described along with results on studies relating to the acquired Ku-band downlink data. It is shown that the errors resulting from the Ku-band downlink through TDRS are characterized by random errors at the demodulator output which give rise to random occurrences of error pairs at the NRZM decoder output. This conclusion is drawn by (1) comparing the empirical error interval distribution from the real EOS data to the theoretical error interval distribution assuming random occurrences of error pairs at the NRZM decoder output, and (2) performing the CVM distribution test on a subset of the real data.

In addition to this, results of a study investigating the performance of the (255,223) Reed-Solomon (RS) code on bursty errors is presented. It is shown that the (255,223) RS code with a depth 5 block interleaver is efficient at correcting bursty errors if the burst durations are less than about 75 binary symbols for a burst duration/rate constant $\nu d \geq 0.01$. Furthermore, comparison of various coding schemes operating on random occurrences of error pairs shows that the (255,223) RS code can achieve the required 10^{-5} decoded error probability with an error probability at the demodulator output of $10^{-2.5}$ using errors-only decoding and $10^{-2.1}$ using erasure decoding.

A brief study dealing with synchronization for the CCSDS transfer frame format which includes a 32 bit PN sequence header shows that achieving synchronization will not be a problem as long as the raw channel error probability is less than about 0.1. Finally, a section is included which gives a brief description of additional capabilities which have been developed for the Communications Link and Error ANalysis (CLEAN) simulation program.

Table of Contents

I. INTRODUCTION

This report describes research performed to date on NASA Grant NAG5-2006 for the period August 16, 1992 through March 30, 1993. This work involves characterizing errors exhibited by the Ku-band downlink through the Tracking and Data Relay Satellite (TDRS) which is to be used for the Earth Observation System (EOS). Also involved in this work is a performance study of various forward error correcting coding schemes on anticipated burst errors.

For the period August 16, 1992 to December 30, 1992, a simulation was developed, called the Communication Link Error ANalysis (CLEAN), to

1) simulate typical errors which may occur in the EOS downlink,
2) simulate/implement various error correcting codes including Reed-Solomon and Convolutional codes,
3) analyze error sequence data.

A description of the work performed can be found in the first semi-annual report [1]. The development of a simulation tailored to the EOS downlink was necessary to investigate issues which do not lend themselves to theoretical analysis and also to be used as a tool to study actual EOS downlink error data acquired by NASA Goddard Space Flight Center (GSFC).

For the period January 1, 1993 to March 30, 1993, the work accomplished includes

1) the continuing development of that simulation,
2) partial results from the error correcting coding study on anticipated (bursty) EOS downlink errors,
3) partial results on the study to characterize the errors resulting from the actual EOS downlink data acquired by NASA/GSFC.
4) results on a study initiated to investigate synchronization issues, including estimates of the average data loss before synchronization is achieved for the actual EOS downlink data.

This work, except for the synchronization study, directly addresses items (A) through (D) in the statement of work for this contract, proposal number 92-3-272, Mississippi State University.

Section II of this report describes the study of the actual EOS downlink data. Section III describes the Ku-band code performance results followed by Section IV which describes the synchronization study. Finally, Section V describes additional developments of CLEAN.

II. TDRS Ku-BAND ERROR CHARACTERISTICS FOR EOS

The Consultative Committee for Space Data Systems (CCSDS) has established a data transfer frame format to be used for space based communication systems [2]. A prototype Wide-band Transfer Frame Formatter (WTFF), used to construct CCSDS transfer frames, was built at NASA/GSFC and used to derive error sequence data for the Ku-band return link through TDRS in June 1992 [3].

A simplified block diagram of the Ku-band return link is shown in Figure 1. The CCSDS transfer frames were sent through the uplink via the Ku-band using one arm of a Quadrature Phase Shift Keyed (QPSK) modulation scheme, and then echoed back to the White Sands Ground Terminal (WSGT) through TDRS. The signal experienced noise and possible Radio Frequency (RF) interference before being received and demodulated at WSGT. The actual data is represented by binary transitions, i.e. a binary "1" is represented by a binary transition, "01" or "10", and a binary "0" is represented by no binary transition, "00" or "11". The NRZM decoder converts the received binary sequence into the data sequence. For the tests conducted, only a Reed-Solomon code was used instead of the CCSDS standard Reed-Solomon/convolutional concatenated code [4].

The error sequence files received represent the errors at the NRZM decoder output. Each received error sequence file was converted to the format required by CLEAN using program EOSCONV [1]. As shown in Table I, a total of 16 error sequence files were received by MSU from NASA/GSFC with error probabilities ranging from roughly 10^{-2} to 10^{-6}. The files range in size from 260kbytes to 33.5Mbytes representing error sequences from $2x10^6$ binary values to $2.6x10^8$ binary values. However, an anomaly in the data, described in Section C below, was discovered after the data was processed. After discussions with Victor Sank at GSFC, it became evident that *the data processed for this report may not be valid.* In any case, we present our findings below for completeness.

To summarize, we found the data to contain random occurrences of error events where each error event consists of two consecutive errors, subsequently called a double error. Since the NRZM decoder will output a double error for each *single* error input, ***we conclude that the channel errors at the demodulator output are random***. However, this does not preclude the possible occurrence of error bursts due to man made or natural RF interference. Therefore in Section III below, we present performance results for various coding schemes over channels in which burst errors occur.

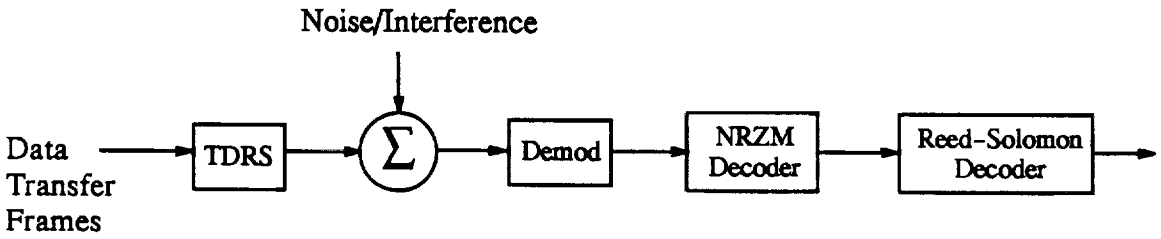


Figure 1. Simplified block diagram of the Ku-band downlink through TDRS.

Table I. Parameters of the Ku-band downlink data received by MSU from GSFC [3] after format conversion using program EOSCONV.

File No.	Original Name	Error Sequence Length	Number of Errors*	Error Density (Bit Error Rate)
1	wser641621.dat	2,087,325	12	$5.7\text{x}10^{-6}$
2	wser641623.dat	268,426,275	1318	$4.9\text{x}10^{-6}$
3	wser641716.dat	2,087,325	16	$7.7\text{x}10^{-6}$
4	wser641721.dat	268,426,275	7556	$2.8\text{x}10^{-5}$
5	wser641837.dat	2,087,325	412	$2.0\text{x}10^{-4}$
6	wser641839.dat	201,314,595	41724	$2.1\text{x}10^{-4}$
7	wser641851.dat	241,966,335	472582	$2.0\text{x}10^{-3}$
8	wser641908.dat	2,087,325	38	$1.8\text{x}10^{-5}$
9	wser641909.dat	134,213,130	2522	$1.9\text{x}10^{-5}$
10	wser641944.dat	268,426,275	567820	$2.1\text{x}10^{-3}$
11	wser642045.dat	268,426,275	136	$5.1\text{x}10^{-7}$
12	wsfler641917.dat	222,034,395	3151969	$1.4\text{x}10^{-2}$
13	wsfler641932.dat	2,087,325	37184	$1.8\text{x}10^{-2}$
14	wsfler641934.dat	134,192,670	2262860	$1.7\text{x}10^{-2}$
15	wsfler642037.dat	2,087,325	50594	$2.4\text{x}10^{-2}$
16	wsfler642038.dat	125,853,600	3654733	$2.9\text{x}10^{-2}$

* Not correct due to a format conversion inherent in FORTRAN

One method of determining burst error statistics, specifically the burst length distribution, is by segmenting the error sequence into error burst regions separated by error free gaps. In Section D, the theoretical burst length distribution using the "gap" segmenting method, is derived for random channel errors. This can be used as an additional test to determine if the channel errors are random.

A. Double-Error Interval Distribution

After displaying several of the error sequences, it became evident that the channel errors at the demodulator output may be random which will result in random occurrences of error pairs at the NRZM decoder output. Two methods were used to determine whether the error events are indeed random. These are described in the following two sections.

1. Cramer Von-Mises Distribution Test

The first method makes use of the Cramer Von-Mises distribution test to determine whether the occurrences of the double errors were indeed Binomial (random). The procedure required that the errors at the demodulator output be determined from the errors at the NRZM decoder output as shown in Figure 1. Each error at the demodulator output corresponds to the occurrence time of a double error at the NRZM decoder output. To this end, an NRZM encoder program was developed (see Section V below) to reverse the effect of the NRZM decoder giving the raw error sequence at the demodulator output. Once the error sequence at the demodulator output is constructed, the Cramer Von-Mises (CVM) distribution test can be applied to see if they are Binomial. The CVM test will determine, with confidence level α, whether the hypothesis that the errors are Binomially distributed (random) can be rejected.

*** PARAGRAPH DELETED ***

Note: An apparent anomaly in the data was due to an incorrect format conversion of the original data due to a program language (FORTRAN) bug. The problem has been fixed but not before the results for this report were completed. The results given below are still valid, however now additional results (not presented here) are possible with the corrected data. All references to data inversions or anomalies in subsequent paragraphs should not be interpreted literally. ***No data inversions exist in the real EOS data obtained from GSFC.***

As a result of an apparent anomaly (which has been since resolved), the CVM test could not be applied to full sequences although some results were obtained by processing partial sequences which did not include the data inversions. The hypothesis for the CVM distribution test is that the errors are indeed Binomial. The CVM distribution test states that the hypothesis should be rejected, with confidence coefficient α, if the test statistic, computed from the error sequence, is greater than the critical value, computed from the theoretical Binomial error distribution given Binomial errors [5]. Critical values for various confidence coefficients, taken from [5], are given here for reference.

α	Critical Value
0.1	0.347
0.05	0.4610
0.025	0.5810
0.01	0.743
0.001	1.168

The confidence coefficient, $\alpha = 0.05$ for example, simply means that if the errors are truly Binomial, then the test statistic will *not* exceed the critical value (hypothesis will *not* be rejected) $(1 - \alpha) \times 100 = 95$ times out of 100.

The CVM test was applied to portions of the NRZM encoded data, corresponding to the data at the demodulator output, using a confidence coefficient $\alpha = 0.05$ with critical value 0.4610. File wsfler642037.dat was truncated to eliminate all the data inversions and the CVM test was applied. The test statistic computed was 0.105 which is less than 0.4610. Therefore, the hypothesis that the errors are Binomial would *not* be rejected. As another case, file wsfler641932.dat was truncated and the CVM test was applied giving a test statistic value of 0.365 which is less than 0.4610. As a final example, the CVM test was applied to the entire wser641837.dat file (no data inversion was found) giving a test statistic value of 0.199. For confidence coefficients 0.05 and below, the hypothesis could not be rejected in all cases. Therefore, *based on these limited results, it is concluded that the double errors occur randomly.*

As a final note, the CVMblk program was applied to those same error segments. Preliminary results suggest that there may be some variation in the error density over small regions on the order of a WTFF but no conclusions have been drawn.

2. Theoretical GAP Distribution

The second method used to demonstrate that the error events occur randomly used the GAP method of segmenting an error sequence, see program GAPest in [1]. The GAPEst program, with GAP = 1, was used to look for consecutive errors in the error sequence. Since the errors at the NRZM decoder output must occur in pairs, the GAPEst program treated each double error as a burst. The GAPEst program outputs the burst interval distribution to a file. If the double errors occur randomly, then the burst interval distribution (each error pair constitutes a burst) must follow the theoretical interval distribution for Binomial errors. Note that it is possible for consecutive channel errors to occur at the demodulator output. When this occurs, error pairs will not be observed at the NRZM decoder output. Although this will bias the burst interval distribution, the probability that this occurs is small so the effect is negligible.

Given that an error occurs, let L_i denote the event that an error free interval of length $i-1$ occurs followed by an error. An error sequence consisting of Binomial (random) errors with error probability p, has a theoretical error interval distribution given by,

$$P\{L_i\} = q^{i-1}p \tag{1}$$

where $q = 1 - p$. Program IntvBin was written to compute this theoretical distribution.

Figure 2 shows the empirical double-error interval distribution for file wsfler642038.dat along with the theoretical Binomial error interval distribution calculated from (1). Clearly the curves match very well. Similarly, Figure 3 shows the empirical double-error interval distribution for file wsfler641932.dat along with the theoretical distribution. Again, the curves match well. Both of these files correspond to error densities at the NRZM decoder output of about $2\text{x}10^{-2}$.

To determine whether the statistics vary with error probability, the empirical double-error interval distribution was computed for files wser1944.dat and wser641851.dat. These are shown in Figure 4(a) and Figure 5(a) respectively. However, the variance of the distribution estimate is quite large due to the fact that relatively few occurrences for each interval length were found (this is the reason for the apparent quantized look of the probability values). Therefore, the distribution was filtered by accumulating every 16 discrete distribution values. These are shown in Figures 4(b) and 5(b), respectively, along with filtered versions of the theoretical distributions. Again, the curves shown match extremely well. As in the previous section, *it is concluded that the double errors occur randomly.*

B. Theoretical GAP distribution for Random Errors

In this section, the theoretical burst length distribution, using the GAP method to segment an error sequence, is considered. It has been shown in the previous section that the errors for the

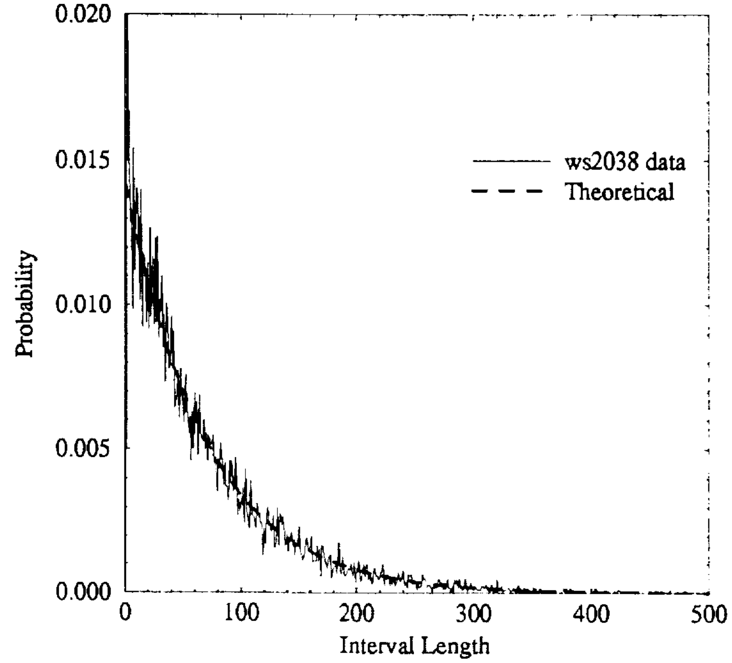


Figure 2. Comparison of the double-error interval distribution for file wsfler642038.dat with the theoretical distribution for Binomial errors.

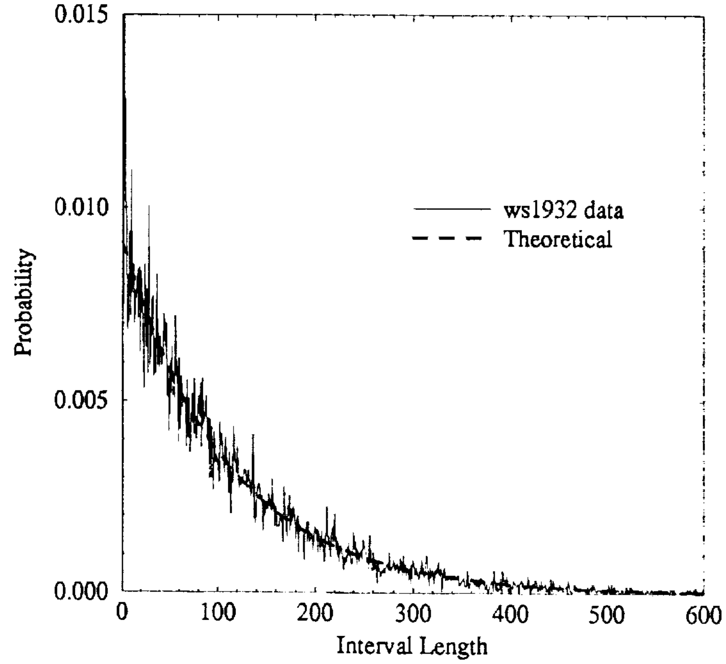
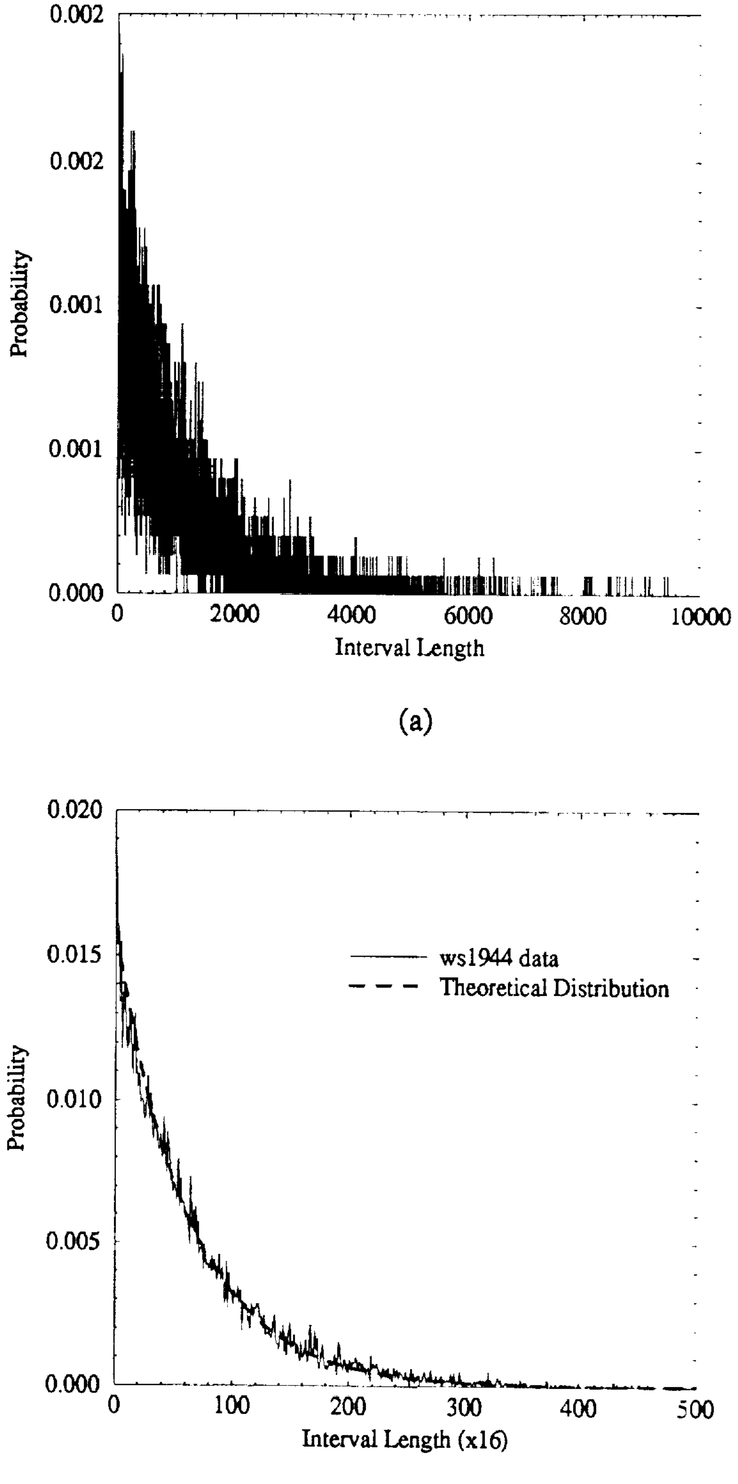


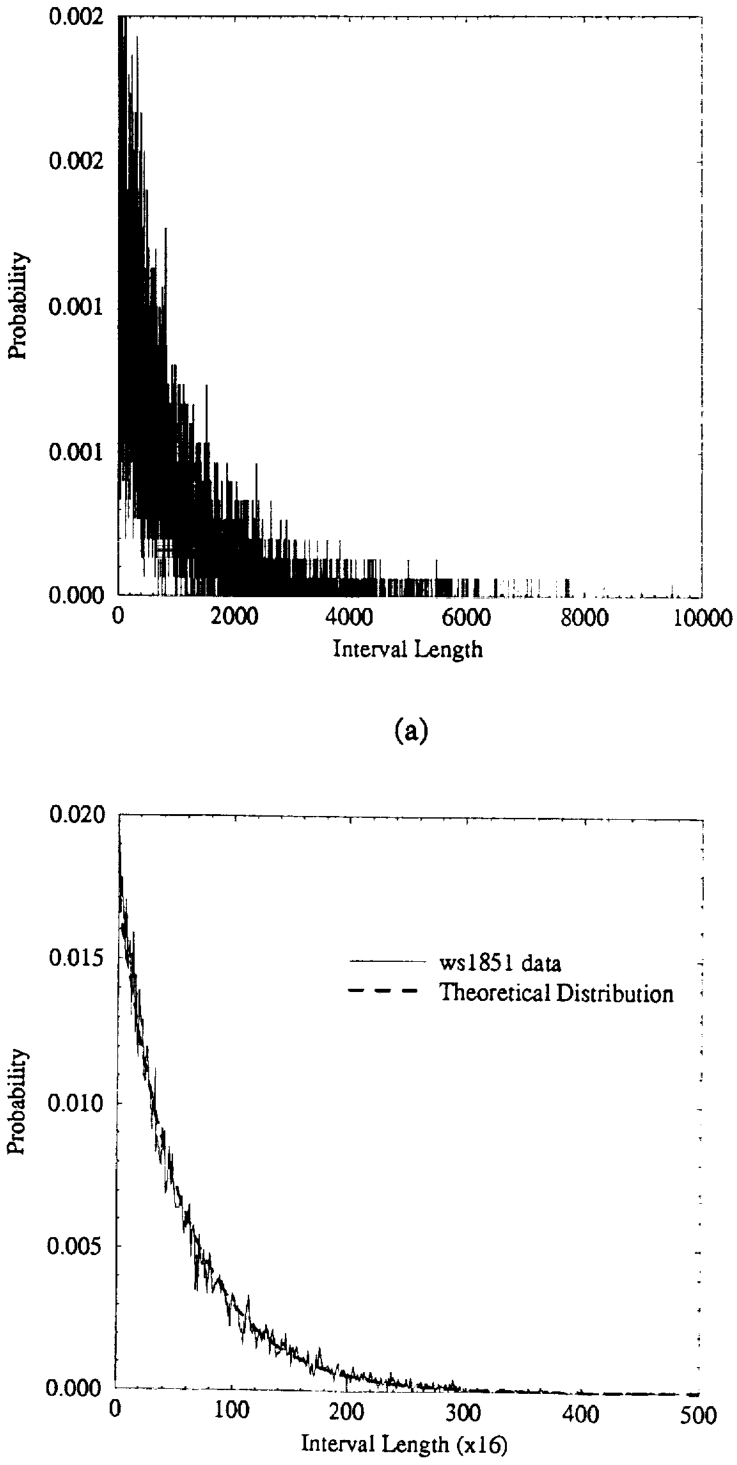
Figure 3. Comparison of the double-error interval distribution for file wsfler641932.dat with the theoretical distribution for Binomial errors.



(a)

(b)

Figure 4. Shown for file wser641944.dat are a) the empirical double-error interval distribution, and b) a comparison of the filtered empirical distribution with the filtered theoretical Binomial-error distribution.



(a)

(b)

Figure 5. Shown for file wser641851.dat are a) the empirical double-error interval distribution, and b) a comparison of the filtered empirical distribution with the filtered theoretical Binomial-error distribution.

Ku-band downlink are random occurrences of double errors. Deriving the exact burst length distribution for these error characteristics is not trivial. However, it is possible to derive the theoretical burst length distribution assuming Binomial (random) errors. Clearly, this theoretical distribution should be somewhat related, perhaps in shape, to the theoretical burst length distribution for random occurrences of double errors. In this section, the theoretical burst length distribution, using the GAP method to segment the error sequence, is derived for Binomial (random) errors.

Suppose that the GAP method for segmenting an error sequence is applied to a sequence of Binomial (random) errors, with error probability p, using a gap length of g. This requires that consecutive error bursts be separated by an error free gap of at least g error sequence values. Let L_i denote the event that an error burst of length i occurs. To compute the probability that L_i occurs, it is assumed that an error free gap has occurred which must be followed by an error. Then $P\{L_i\}$ is the probability that a burst of length i occurs followed by an error free gap (which satisfies the assumption for the next burst occurrence). Three cases are considered separately.

First consider event L_1, which represents an error burst of length 1. Given that a gap has occurred (which must end with an error), L_1 will occur if an error free gap of length g occurs. Since the probability that an error occurs is assumed to be p, we have

$$P\{L_1\} = (1-p)^g = q^g \tag{2}$$

where $q = 1-p$ is the probability that a correct error sequence value occurs.

Second consider the event L_i for $2 \le i \le g+1$. Since each error burst must begin and end with an error, these errors are referred to as the terminal errors of the burst. Let the error sequence values for event L_i be assigned labels s_1, s_2, s_3, ..., s_i as shown in Figure 6. The terminal errors for the burst correspond to s_1 and s_i. Since a burst of length $g+1$ has $g-1$ error sequence values between the two terminal errors, it is not possible for a gap of length g to occur within the error burst. Therefore, the probability that a burst of length $2 \le i \le g+1$ occurs is only a function of the probability that an error occurs for error sequence value s_i followed by an error free gap of length g. This gives,

$$P\{L_i\} = pq^g \quad , \quad 2 \le i \le g+1 \tag{3}$$

Note that this probability is not a function of the burst length.

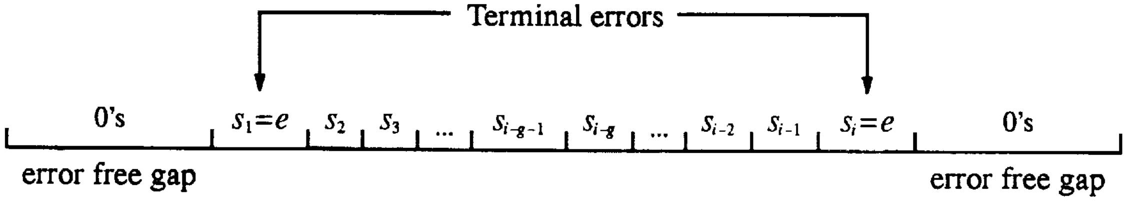


Figure 6. Illustration of the error sequence values for an error burst.

Third consider the event L_i for $i \geq g+2$ (see if this is the same as the derivation in the handwritten write-up). For a burst of length i to occur, the burst must be composed of sufficient errors between the two terminal errors so that no gap of length g occurs, otherwise the segmentation process would break those errors into two bursts. Let W_j denote the event that no error free gap of length g occurs for the error sequence values s_2 through s_j. With this definition, it is possible to express the burst length probability as

$$P\{L_i\} = P\{W_{i-1}\}\, pq^g \quad , \quad i \geq g+2 \tag{4}$$

It is only necessary to specify $P\{W_{i-1}\}$.

Although no closed form expression for $P\{W_{i-1}\}$ has been found, a recursion formula with known initial conditions is derived below which completely specifies $P\{W_{i-1}\}$. Let e_j denote the event that s_j is in error and let $\overline{e}_j$ denote the event that s_j is not in error. Consider the following cases:

Case 1: e_{i-1} occurs. For this case, event W_{i-1} will occur if W_{i-2} occurs. The probability that e_{i-1} occurs is given by p.

Case 2: e_{i-2} and $\overline{e}_{i-1}$ occur. For this case, event W_{i-1} will occur if W_{i-3} occurs. The probability that e_{i-2} and $\overline{e}_{i-1}$ occur simultaneously is given by pq.

Case 3: e_{i-3}, $\overline{e}_{i-2}$, $\overline{e}_{i-1}$ occur. For this case, event W_{i-1} will occur if W_{i-4} occurs. The probability that e_{i-3}, $\overline{e}_{i-2}$, and $\overline{e}_{i-1}$ occur simultaneously is given by pq^2.

The cases to be considered continue in a like fashion until we have,

Case $g-1$: e_{i-g+1}, $\overline{e}_{i-g+2}$, $\cdots$, $\overline{e}_{i-1}$ occur. For this case, event W_{i-1} will occur if W_{i-g} occurs. The probability that e_{i-g+1}, $\overline{e}_{i-g+2}$, $\cdots$, $\overline{e}_{i-1}$ occur simultaneously is given by pq^{g-2}.

Case g: e_{i-g}, $\overline{e}_{i-g+1}$, $\cdots$, $\overline{e}_{i-1}$ occur. For this case, event W_{i-1} will occur if W_{i-g-1} occurs. The probability that e_{i-g}, $\overline{e}_{i-g+1}$, $\cdots$, $\overline{e}_{i-1}$ occur simultaneously is given by pq^{g-1}.

Now observe that the events described in cases 1 through g above are mutually exclusive. That is, an error burst pattern of length i must fall under one of the cases identified above and yet no error burst pattern can fall under two or more of the cases simultaneously. Therefore, by the theorem of total probability, $P\{W_{i-1}\}$ can be expressed as

$$P\{W_{i-1}\} = \sum_{j=0}^{g-1} P\{W_{i-j-2}\}\, pq^{j}\,, \quad i \geq g+2 \tag{5}$$

The iteration is initialized by defining,

$$P\{W_{i-1}\} = \begin{cases} 0 & i < 1 \\ 1 & 1 \leq i \leq g+1 \end{cases} \tag{6}$$

Note that when $i = g+2$, there are exactly g binary symbols between the terminal errors. The only way the GAP segmenting process would split the terminal errors is if all the symbols between the terminal errors were error free. Clearly, the probability that this occurs is q^{g} which means $P\{W_{g+1}\}$ must equal $1-q^{g}$. It can be shown that $P\{W_{g+1}\}$, evaluated using (5), reduces to $1-q^{g}$.

In this section, the burst length distribution resulting from the GAP segmentation process for random errors has been derived. The probability that a burst of length i occurs is $P\{L_i\}$ given by (2), (3), and (4). To compute $P\{L_i\}$ for $i \geq g+2$, (5) must be evaluated with initial conditions given by (6). Program BinGAP performs these calculations.

To verify the derivation, CLEAN was used to generate the empirical burst length distribution, by executing the GAP segmentation process, on synthetically generated Binomially distributed errors. Figure 7 shows close agreement between the theoretical burst length distribution and the empirical distribution for Binomial errors with error probability of 0.05 and GAP length of 6.

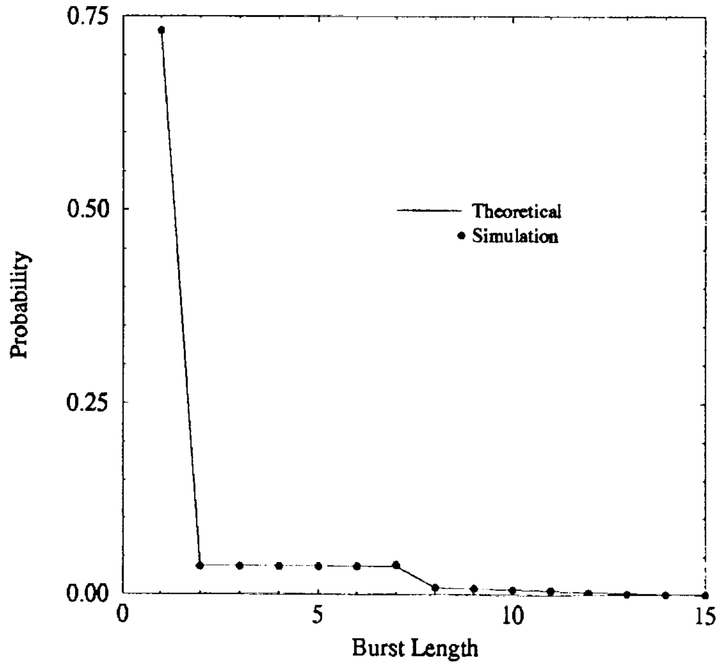


Figure 7. Comparison between the theoretical burst length distribution, using the GAP segmentation method, and the empirical distribution for Binomial errors with error probability 0.05 and GAP length of 6.

III. TDRS Ku-BAND CODE PERFORMANCE

An efficient coding/interleaving scheme is one which provides the required performance, such as 10^{-5} bit-error probability or perhaps 2db coding gain, with minimal overhead (maximum information throughput) and with minimal delay. All error correcting codes provide a performance gain, in the form of a reduced error probability and/or in the form of a reduced required transmitter power. However, when an error correcting code is inserted into an existing data link, this gain is achieved at the expense of information rate throughput. The reduction in information throughput when an error correcting code in inserted in an existing system is identically equal to the code rate.

For typical space communication links, it is required that the downlink provide a 10^{-5} error probability. Therefore given this requirement, the problem becomes one of selecting an error correcting code which provides the required error probability with minimum code rate. The CCSDS concatenated coding standard consists of a rate 1/2 convolutional inner code and a (255,223) RS outer code. The net concatenated code rate is (1/2)(223/255) = 0.437. Although the concatenated code will undoubtedly achieve the required error probability, it is not necessarily the best code choice for EOS which is to use the Ku-band downlink through TDRS.

In Section A that follows, the performance of the (255,223) RS code operating on burst errors is considered. In Section B, a comparison of various coding schemes including the CCSDS concatenated code is considered on typical errors occurring at the NRZM decoder output as characterized in Section II above.

Before proceeding, an issue of concern is the reliability of decoded error probability estimates obtained by the simulation. Preliminary work indicated that the simulation run length, required to achieve small variance estimates of the decoded error probability for a (255,223) Reed-Solomon coded system, must be very long. Background work has indicated that the decoder must output a large number of errors, on the order of many hundreds, before a small variance estimate of the decoded error probability is achieved. This background work culminated in a paper which has been submitted to the IEEE Military Communications Conference, 1993 [6].

A. Burst Duration Study for a Reed-Solomon Code

The (255,223) RS code has a code rate of (223/255) = 0.875 which is exactly double the code rate of the concatenated code. Therefore, eliminating the convolutional inner code immediately provides a factor of 2 increase in the information rate throughput of the system. However, this is acceptable only if the decoded error probability meets the required 10^{-5}, or lower, error probability specification. A proposed coding scheme for the EOS downlink consists of (255,223) RS encoded data frames multiplexed in the form of a CCSDS transfer frame. A CCSDS transfer frame consists of 5 multiplexed RS encoded data streams which provides the effect of a depth 5 block interleaver. Therefore, the system under study in this section consists of a (255,223) RS code used with a depth 5 byte-oriented block interleaver. Note that the following results do not incorporate the use of NRZM data. The results in this section are intended to demonstrate the efficiency with which RS codes can correct burst errors. For the final report, results will be shown for bursty errors which incorporate the use of NRZM data.

It is well known that space based communication systems operate over channels which exhibit random errors along with error bursts. Therefore, in this section, we consider the performance of the proposed coding scheme operating on burst errors. Burst errors can be characterized by four distributions/parameters: burst duration distribution (d), rate of burst occurrence (ν), error probability within the bursts (P_{eb}), and error probability outside the bursts (P_{eg}). The following analysis consists of two cases considered below.

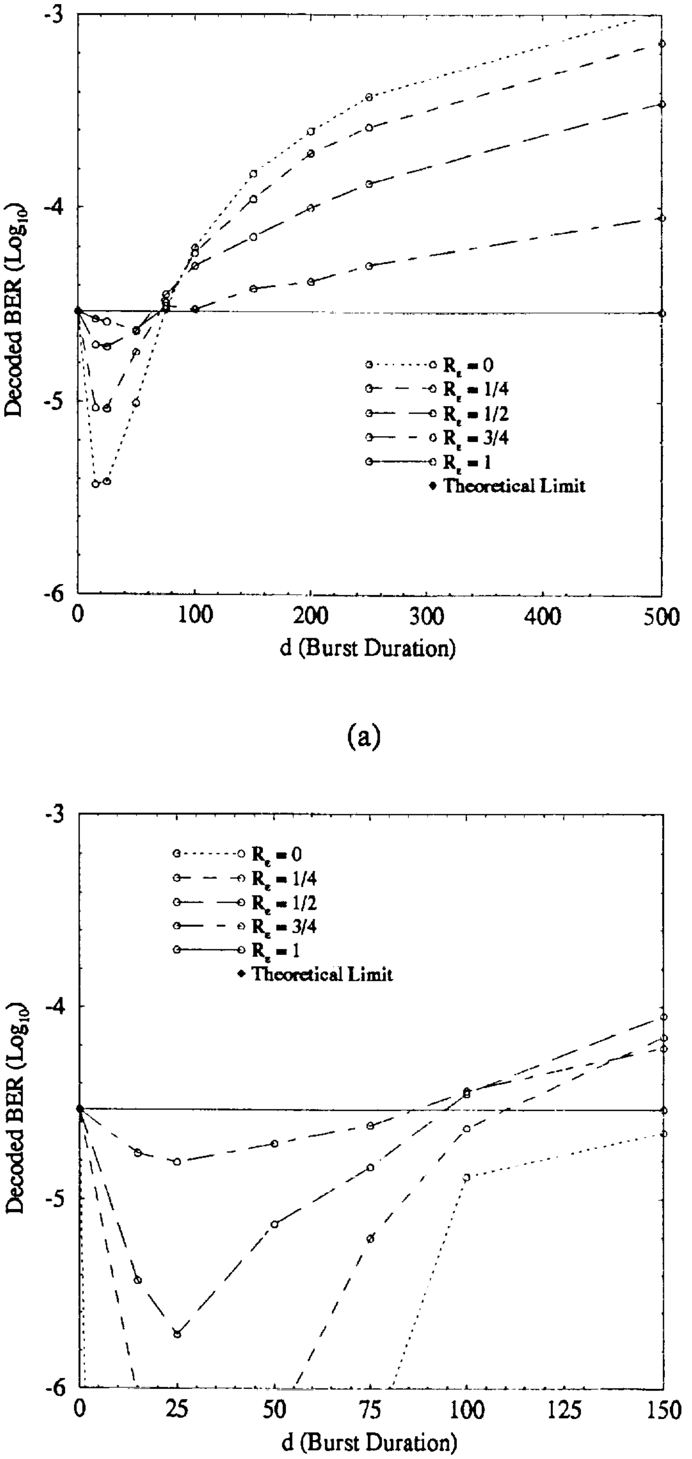
For the first case, it is assumed that error bursts occur so infrequently that only one error burst will occur within a single interleaved block. In this case, the required error probability will be achieved if the maximum anticipated error burst can be corrected. A (255,223) RS code can correct a maximum of $t = 16$ code symbol errors or $(16)(8) = 128$ binary symbol errors. Since an error burst is split into 5 segments by the depth 5 block interleaver, a total error burst of length $(128)(5) = 640$ is correctable provided no other errors occur within the interleaved block. For a 75Mbps channel rate, this corresponds to an RFI interference burst of 8.5µsec in duration. For a 150Mbps channel rate, corresponding to the high data rate requirement, this corresponds to an RFI interference burst of 4.3µsec in duration.

However, it is probable that random errors will occur within an interleaved block at the same time that an error burst occurs. To consider the worst case, suppose that the binary channel error probability, excluding error bursts, is 0.005. Each interleaved block contains a total of $(255)(8)(5) = 10,200$ binary symbols (10,200/8 = 1275 code symbols) which means that an average of $(0.005)(10,200) = 51$ binary symbol errors will occur within each interleaved block. These binary symbol errors will cause $1275[1-(1-0.005)^8] \approx 50$ code symbol errors [7]. Therefore, each interleaved block can correct, on average, an error burst of length $8[(16)(5)-50] = 240$ binary symbol errors. This corresponds to an RFI burst of length 3.2µsec and 1.6µsec in duration for the current requirement and for the high data rate requirement, respectively. The following is a table giving the average correctable burst lengths, in binary symbol errors, for different values of the binary channel error probability (excluding error bursts).

Binary Channel Error Probability	Average Maximum Correctable Error Burst
0.01	Zero
0.005	240
0.001	560
0.0005	600
0.0001	632

For the second case, it is assumed that error bursts occur frequently enough that the probability of multiple bursts occurring per interleaved block cannot be ignored. For this case, performance is strongly related to the burst duration, burst rate product νd, the ratio of random errors to burst errors R_ε, and the channel error probability. Figure 8(a) shows the decoded error probability, called the Bit-Error Rate (BER), for a (255,223) RS coded system with a depth 5 block interleaver, with $\nu d = 0.05$, and with a channel error probability of $P_{ch} = 0.004$. The channel error probability is the error density at the demodulator output due to all errors whether they are caused by thermal noise or RFI. Clearly, for burst durations less than 75 binary symbols, *performance increases as R_ε, the fraction of random errors to total errors, decreases.* This is due to the fact that an RS code is a byte oriented code and demonstrates the efficiency with which an RS code can correct bursty errors. However, for burst durations greater than 75 binary symbols, performance decreases as R_ε decreases due to the fact that each error burst which fills the interleaved block causes decoding failure for the 5 RS codewords contained within that interleaved block. The result is a decrease in system performance.

Figure 8(b) shows the decoded BER for the same parameters used for Figure 8(a) but with $\nu d = 0.01$. Note the scale change for the abscissa. Although the curves do not all cross at a common point, it still appears that performance increases with decreasing R_ε for burst durations less than 75 binary symbols. Indeed, the achievable error probability for $R_\varepsilon = 0$ (all errors occur in bursts) is much below the required 10^{-5} for all burst durations below 75 binary symbols. In fact, the achievable error probability for $R_\varepsilon = 1/4$ is below the required 10^{-5} for all burst durations



(a)

(b)

Figure 8. Performance of a (255,223) Reed-Solomon code with a depth 5 block interleaver operating on bursty errors with $P_{ch} = 0.004$ and with (a) $\nu d = 0.05$, and (b) $\nu d = 0.01$.

below 75 binary symbols. Using obvious decoded BER trends as a function of defined parameters, the conclusion can be drawn that the decoded BER achieves the required 10^{-5} error probability when the following conditions are met:

a) $d < 75$ binary symbols

b) $\nu d < 0.01$

c) $P_{ch} < 0.004$

If these conditions are not met, then the decoded BER may or may not achieve the required BER, however the following trends are noted:

a) Decoded BER decreases with decreasing R_{ε} assuming d less than some threshold

b) Decoded BER decreases with decreasing νd

c) Decoded BER decreases with decreasing P_{ch}

d) Decoded BER decreases with increasing interleave depth

More results will be provided in the final report.

B. Comparison of Coding Schemes

In Section II above, it is concluded that the errors at the demodulator output are random which gives random occurrences of error pairs at the NRZM decoder output. In this brief section, a comparison of various coding schemes operating on the types of errors anticipated at the NRZM decoder output is given.

Figure 9 shows the decoded BER as a function of the *raw error probability at the demodulator output* for various coding schemes including a (255,223) RS code, rate 1/2 constraint length 7 convolutional code (CCSDS standard inner code) without a Periodic Convolutional Interleaver (PCI), the same convolutional code with a PCI, the CCSDS standard concatenated code, and a stand alone (255,223) RS code using erasure decoding. The results indicate that the concatenated code performed the best. However, again it is emphasized that the best code choice is the one with the lowest code rate and complexity that meets the 10^{-5} specification (or other specification). The following is a summary of the raw error probability at the demodulator output required to give a decoded BER of 10^{-5} for the various coding schemes.

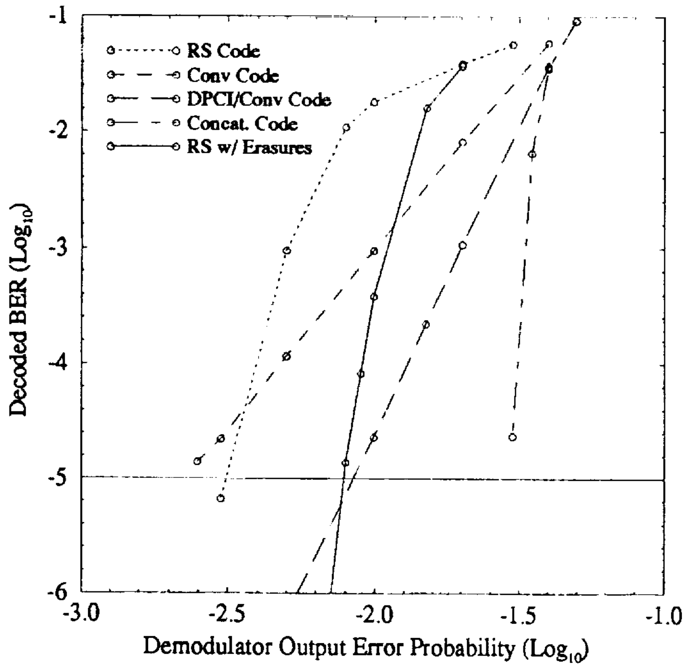


Figure 9. Comparison of various coding schemes assuming random errors at the demodulator output. These results assume that an NRZM decoder follows the demodulator so that the errors input to the error correcting decoder are random occurrences of error pairs.

Code Type	Demod Output BER required to achieve 10^{-5}
CCSDS Concatenated	$10^{-1.5}$
Convolutional, PCI	$10^{-2.08}$
(255,223) RS (Erasure decoding) Depth 5 Block Intlv	$10^{-2.1}$
(255,223) RS (Errors only decoder) Depth 5 Block Intlv	$10^{-2.5}$
Convolutional, no PCI	$10^{-2.65}$

If it is possible to achieve at $10^{-2.5}$ BER at the demodulator output, then the best code choice is the (255,223) RS code. This code has a higher code rate than either the convolutional code or the concatenated code. The (255,223) RS code is also efficient at correcting error bursts as shown in Section A above. However, to conclude that the (255,223) RS code is the best choice requires that performance be investigated for bursty errors which would occur at the NRZM decoder output. More results will be given in the final report.

Note that, as shown in Figure 9, the (255,223) RS code used with erasure decoding performs as well as the convolutional code with the PCI for decoded error probabilities of 10^{-5} and below. It is prudent to use erasure RS decoding when RFI is present in the channel due to the fact that erasure decoding can correct, at best, twice as many errors per codeword than an errors only decoder. It is possible to predict code symbol errors at the decoder input which are likely to be in error by investigating soft decision values output by the demodulator [8].

IV. SYNCHRONIZATION

There are two fundamentally different facets of a communication system which relate to user data at the receiver; synchronization and steady state operation. Synchronization is the process by which the receiver attempts to extract timing information from the received signal so that optimum demodulation of the transmitted data can be achieved. While synchronization is attempted, data transmitted across the link is lost. This gives rise to highly unreliable and therefore unusable data to the user. Once synchronization has been achieved, steady state operation of the receiver provides highly reliable, although not error free, data to the user which is usable.

If the communication system is viewed as a system which processes binary data at the input to give nearly identical binary data at the output, then synchronization would correspond to the transient portion of the system and the steady state error operation (once synchronization has been achieved) would correspond to the steady state response of the system. Although these two facets of the system are related, the data output by each is treated differently, one gives rise to unusable data (synchronization) and one gives rise to usable data (steady state operation), which allows them to be analyzed separately. Steady state error statistics has been investigated in Sections II and III above. In this section, synchronization is considered.

Although synchronization issues were not a formal part of the statement of work for this contract, the investigators feel that it is an important issue which must be investigated in conjunction with steady state error statistic studies.

The CCSDS recommendation for Packet Telemetry [2] requires that a 32 bit PN sequence header precede each CCSDS transfer frame. Synchronization is said to be achieved when two consecutive 32 bit PN sequence headers are correctly detected. In the following development, the average number of frames lost before achieving synchronization is derived for random channel errors. This provides a measure of how much data would be lost, on average, before achieving synchronization for the Ku-band downlink.

To achieve synchronization, a 32 bit window is applied to the hard decision data at the demodulator output. Detection of the 32 bit PN sequence pattern is assumed to occur when the Hamming distance between the transmitted PN sequence header and a 32 bit pattern at the receiver is less than or equal to the preselected threshold, Q. In other words, detection occurs when at least 32-Q of the 32 bits match the original PN sequence pattern. For each 32 bit pattern considered, one of two possibilities exist. Let H_0 denote the event that the 32 bit pattern corresponds to the transmission of the 32 bit sequence header and let H_1 denote the event that the 32 bit pattern does ***not*** correspond to the transmission of the 32 bit sequence header. If detection of the 32 bit sequence header occurs given H_0, then the 32 bit sequence header is correctly detected.

Let the probability that this event occurs be denoted P_d. If detection of the 32 bit sequence header occurs given H_1, then false detection occurs. Let the probability that this event occurs be denoted P_{fd}.

If there are no channel errors, then the PN sequence header would be detected every time, each detection spaced exactly 10,200 binary symbols apart corresponding to the 5 RS codeblocks of data which follows each header. Synchronization is said to be achieved when two consecutive PN sequence headers are detected by the correct binary symbol spacing.

Suppose the channel errors are random and occur with probability P_{ch}. Then the distribution of errors within each 32 bit PN sequence header at the receiver is given by a Binomial distribution. Let h_i denote the event that i binary symbol errors occur given H_0. Then the probability that h_i occurs is given by

$$P\{h_i\} = \binom{32}{i} P_{ch}^{i} (1 - P_{ch})^{32-i} \tag{7}$$

If the threshold is used to declare detection of the 32 bit PN sequence header, then detection will occur when h_i for $i \le Q$ occurs given H_0. That is,

$$P\{P_d \mid H_0\} = \sum_{i=0}^{Q} P\{h_i\} \tag{8}$$

If it is assumed that the binary symbols corresponding to the data are completely random (and the PN sequence header is delta correlated), then the probability of false detection is easily shown to be

$$P\{P_{fd} \mid H_1\} = \sum_{i=0}^{Q} \binom{32}{i} \left(\frac{1}{2}\right)^{32} \tag{9}$$

Since there are 10,200 binary data symbols per frame, there are 10,200, 32 bit sequences per frame which can result in false detection. Therefore, the probability that false detection occurs in a given frame is $(10,200)P\{P_{fd} \mid H_1\}$. For $P_{ch} < 0.1$, it is easily shown that $(10,200)P\{P_{fd} \mid H_1\} \ll P\{P_d \mid H_0\}$. So (7) and (8) define the probability that the 32 bit PN sequence header will be correctly detected using the simple thresholding technique with threshold Q.

Synchronization is said to be achieved when two consecutive PN sequence header detections occur which are spaced apart by the proper number of binary symbols. The average number of frames lost will be the average number of missed PN sequence headers before two consecutive detections occur. This problem is not unlike that of determining the theoretical burst length distribution described in Section II.A.2 above. For the moment, view each received

CCSDS transfer frame as a single entity. Consider generating a *frame detection error sequence* by substituting a single error symbol in a newly generated error sequence for each frame received. A binary 0 in the frame detection error sequence is used to denote the correct detection of the 32 bit PN sequence header. A binary 1 in the frame detection error sequence is used to denote a missed detection of the 32 bit PN sequence header. Then synchronization is achieved when two consecutive binary zeros occur in the frame detection error sequence because this corresponds to two consecutive correct detections of the 32 bit PN sequence header. The average number of binary symbols passed before two consecutive zeros occur corresponds to the number of missed frames before achieving synchronization. This average can be found by generating the frame detection error sequence, applying the GAP error segmentation method, and determining the average burst length. The results of Section II.A.2 can be used directly to determine the burst length distribution using the probability of PN sequence header detection $P\{P_d \mid H_0\}$ in place of the channel error probability, p, used in the development in Section II.A.2.

Let $p = 1 - P\{P_d \mid H_0\}$, $q = 1 - P\{P_d \mid H_0\}$, and let L_i denote the probability that i frames are past before achieving synchronization, i.e. two consecutive correct 32 bit PN sequence header detections. Then the probability that L_i occurs for $g = 2$ is given by (2) through (6) to be

$$P\{L_i\} = \begin{cases} q^2, & i = 1 \\ pq^2, & 2 \le i \le 3 \\ P\{W_{i-1}\}pq^2, & i \ge 4 \end{cases} \tag{10}$$

where

$$P\{W_{i-1}\} = \sum_{j=0}^{1} P\{W_{i-j-2}\}\, pq^j, \quad i \ge 4 \tag{11}$$

and where the iteration is initialized by

$$P\{W_{i-1}\} = \begin{cases} 0 & i < 1 \\ 1 & 1 \le i \le 3 \end{cases} \tag{12}$$

Therefore, the probability that i frames are lost before achieving synchronization is given by (10) along with (11) and (12).

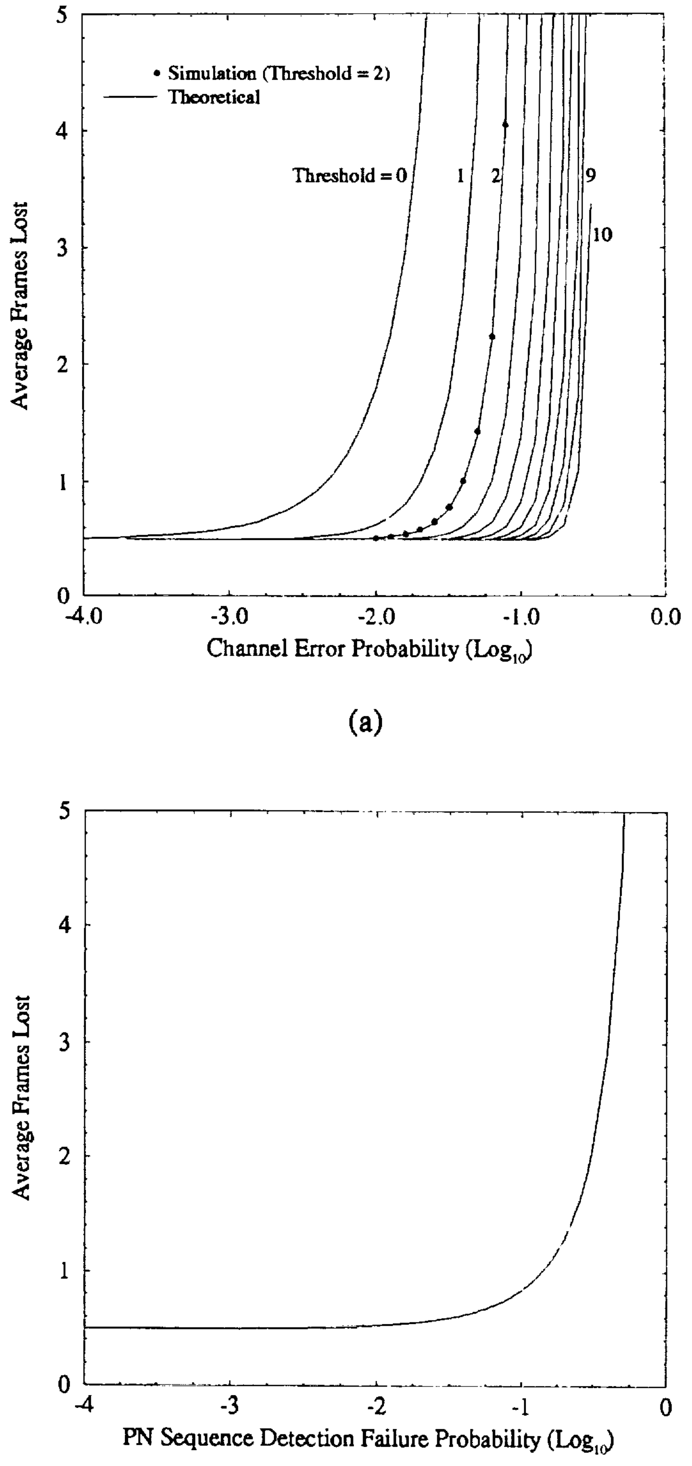
To establish the average number of frames lost before synchronization occurs, consider the fact that the starting point for the search to detect the 32 bit PN sequence headers can start anywhere within a frame. If it is assumed that the starting point is equally likely to be anywhere within the first frame, then the average number of frames lost will be 1/2 plus the average frames lost after the first PN sequence header is encountered. Using this along with (10), the average number of frames lost before synchronization is achieved, denoted F, can be determined. Note that it is assumed that the data between the two correctly detected PN sequence headers is usable data and therefore does not constitute a lost frame. Given this we have

$$F = \frac{1}{2} + \sum_{i=1}^{\infty} i\, P\{L_i\} \tag{13}$$

where the 1/2 accounts for the uniformly distributed starting point within a frame. Note that the average data loss, in binary symbols, can be found by simply multiplying F by the data length in each frame. Program SyncPb was written to calculate F for a given channel error probability P_{ch}.

Figure 10(a) shows the average number of frames lost before achieving synchronization as a function of the channel error probability assuming that the channel errors used to detect the PN sequence are independent. The figure shows that simulation results for a threshold of $Q = 2$ match very well with the theory. Figure 10(b) shows the average number of frames lost before achieving synchronization as a function of the PN header detection failure probability. This result shows that as long as the PN header detection failure is less than about $10^{-3/4} = 0.18$, synchronization will be achieved within a frame or two. Program 'Sync' has been written to measure the average frames lost for an actual error sequence input to the program.

The Sync program was used to estimate the average number of frames lost for each of the real EOS data files received from GSFC. The results show that synchronization is not a problem for any of the real error sequences. The worse case arose for file wser642038.dat which yields an average of 0.6 frames lost before achieving synchronization.



(a)

(b)

Figure 10. Average frames lost as a function of a) channel error probability assuming independent channel errors, and b) PN sequence header detection failure probability.

V. FURTHER DEVELOPMENTS FOR CLEAN

This section briefly describes additional capabilities which have been added to CLEAN. The capabilities have been divided into two main sections. In Section A, additional error sequence manipulation programs, which represent system components, are briefly described and in Section B, programs written to evaluate theoretical formulas are briefly described.

A. System Component Program Modules

To more accurately represent the real EOS Ku-band downlink and to manipulate error sequences, several new programs were written.

1. NRZMEncd (NRZM Encoder)

This program performs differential (NRZM) encoding on the error sequence. That is, the encoder output toggles (changes binary status) when the input bit is a binary "1" and remains the same when the input is a binary "0". The program reads in the error sequence by block and performs differential encoding on each block and then writes the modified block back out to the error.seq file.

Executing the program causes the 'error.seq' file to be read which contains an error sequence (in packed format). The 'error.seq' file must exist prior to the execution of this program. There are no assumptions associated with the implementation or output of this program.

2. NRZMDec (NRZM Decoding)

This program performs differential (NRZM) decoding on the error sequence. That is, the decoder outputs a binary "1" when a transition occurs on the input data stream (a "0" followed by a "1" or a "1" followed by a "0"). A binary "0" is output when no transition occurs (a "0" followed by a "0" or a "1" followed by a "1"). The input error sequence is assumed to be stored in default file 'error.seq' and the differentially decoded data stream is also output to that file.

The program reads in the error sequence by blocks and performs differential decoding on each block and then writes the modified block back out to the error.seq file. The program outputs several statistics to the user screen as well.

Executing the program causes the 'error.seq' file to be read which contains an error sequence (in packed format). The 'error.seq' file must exist prior to the execution of this program. There are no assumptions associated with the implementation or output of this program.

3. Sync (Synchronization)

This program calculates the synchronization probability for a synchronization scheme in which each data frame is preceded by a PN sequence header. Synchronization will be said to occur when two consecutive PN sequences have a Hamming distance less than or equal to the Threshold. The synchronization probability is calculated by considering all consecutive pairs of PN sequences in the error sequence and counting the percentage which would achieve synchronization.

This program inputs parameters from an ASCII data file with default name 'Sync.prm' and inputs the error sequence from data file with default name 'error.seq'. Various statistics are output to an ASCII data file with default name 'Sync.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

The program is run by editing the parameter file 'Sync.prm' and selecting the appropriate parameters and by choosing a program ID by editing file 'ID.prm'. The 'error.seq' file must exist prior to the execution of this program.

4. DisplSeg (Display a Segment of the error sequence)

This program displays a segment of the error sequence found in file 'error.seq'. It is assumed that the errors stored in error.seq are in the DBESS (Double Byte Error Sequence Symbol) packed format. The program is interactive and asks the user to input the starting address of the segment to be displayed as well as the segment length.

5. SvHist (Save Histogram as a pdf)

This subroutine writes out a histogram array, as a probability density function (pdf), to a file in real*8 format. The unit attached to the opened file is 10. At the time that this subroutine is called, unit=10 must not be
assigned.

This subroutine also outputs statistics of the pdf to the user screen and also to the log file assumed to be open as unit=8.

6. SeqTrunc (Sequence Truncation)

This program truncates an error sequence in length by modifying the value of N stored in the error sequence file header. This program is meant to be fast (the truncation only requires that a single value in the error seq be modified) but this method is clearly memory inefficient due to the fact that the error.seq file size remains unchanged.

7. ASCII (ASCII conversion program)

This program converts a pdf, stored as real*8, to an ascii format. The ascii output file has a header which identifies the index and value columns followed by a series of records of the form X Y where X is the index value and Y is the sequence (pdf) value.

This program will convert individual files or can convert a series of data files and combine them into a single ASCII file. There are no assumptions associated with the implementation or output of this program.

B. Theory related programs

Several programs have been written to evaluate the theoretical formulas discussed in previous sections of this report.

1. IntvBin (Theoretical PDF for Binomial Errors)

This program calculates the theoretical error interval probability density function for an error sequence in which the errors are independent. This calculation is easily derived but is also documented in, Kenneth Brayer, "Error Patterns Measured on Transequitorial HF Communication Links", IEEE Trans on Comm Tech, Vol COM-16, No. 2, April 1968, p. 216. The probability of getting an *n*-bit gap is:

$$P\{C^n e \mid e\} = \sum_{k=0}^{n} p(1-p)^k$$

The program outputs the pdf to file 'IntvBin.pdf' which is a direct access file where each real*8 pdf value is stored per record.

Note that there are NO parameters to be read in for this program. However, various statistics are output to an ASCII data file with default name 'IntvBin.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'. There are no assumptions associated with the implementation or output of this program.

2. BinGAP (Binomial error GAP distribution)

This program calculates the theoretical GAP burst length distribution for Binomially distributed errors which is described in Section II.B above. The final distribution is output to file 'BinGAP.pdf' as a pdf file. This program inputs the GAP parameter from the user interactively. Various statistics are output to an ASCII data file with default name 'BinGAP.log'.

3. SyncPb (Synchronization with the Channel error probability)

This program calculates the synchronization probability for a synchronization scheme in which each data frame is preceded by a PN sequence header. Synchronization will be said to occur when two consecutive PN sequences have a Hamming distance less than or equal to the Threshold. The synchronization probability is calculated by considering all consecutive pairs of PN sequences in the error sequence and counting the percentage which would achieve synchronization.

It is assumed that the binary channel errors are independent. This program inputs parameters from the user interactively. Various statistics are output to an ASCII data file with default name 'SyncPb.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

4. SyncPPN (Synchronization with probability of PN detection failure)

This program calculates the synchronization probability for a synchronization scheme in which each data frame is preceded by a PN sequence header. Synchronization will be said to occur when two consecutive PN sequences have a Hamming distance less than or equal to the Threshold. The synchronization probability is calculated by considering all consecutive pairs of PN sequences in the error sequence and counting the percentage which would achieve synchronization.

This program inputs parameters from the user interactively. Various statistics are output to an ASCII data file with default name 'SyncPPN.ID', where ID is a three letter identifier for the current run which is input from file 'ID.prm'.

5. QuantPDF (Quantize PDF)

This program inputs a pdf from a file and quantizes it into ranges specified by the user. The values for the Quantized PDF are found by integrating (summing) the PDF in the specified quantization ranges. The PDF is read in from a file assumed to be stored in direct access format.

There are no assumptions associated with the implementation or output of this program.

BIBLIOGRAPHY

1. Ebel, W.J., and Ingels, F.M., An Investigation of Error Characteristics and Coding Performance, MSU Department of Electrical and Computer Engineering, Technical Semi-Annual Report, December 30, 1993, NASA Grant NAG5-2006.

2. Packet Telemetry, Recommendation CCSDS 102.0-B-2, Issue 2, Blue Book, Consultative Committee for Space Data Systems, January 1987.

3. Sank, V.J., and Conaway, W.H., WTFF Test Report, Technical Report, Fredrick Herold & Associates, Inc., March 10, 1993, Draft.

4. Telemetry Channel Coding, Recommendation CCSDS 101.0-B-2, Issue 2, Blue Book, Consultative Committee for Space Data Systems, January 1987.

5. Kendall, M.G., and Stuart, A., The Advanced Theory of Statistics, Vol. 2, Hafner Publishing Co., New York, 1973.

6. Ebel, W.J., and Ingels, F.M., "Confidence Intervals for Simulations using Reed-Solomon Codes", IEEE Military Communications Conference, 1993, submitted.

7. Ebel, W.J., Simulation and Evaluation of Reed-Solomon Codes in a Burst Noise Environment, Ph.D. Dissertation, University of Missouri-Rolla, 1991.

8. Ebel, W.J., and Ingels, F.M., Reed-Solomon Erasure Decoding Using Channel Measurement Statistics, Unsolicited Proposal No. 93-3-362, Mississippi State University, February 1993.

APPENDIX III

An Investigation of Error Characteristics and Coding Performance

NASA GRANT NAG5-2006

Annual Report

January 1993 - August 1993

AN INVESTIGATION
OF
ERROR CHARACTERISTICS
AND
CODING PERFORMANCE

by

Will Ebel
and
Frank Ingels

Mississippi State University

GSFC
Greenbelt, MD
7/20/1993

OBJECTIVES

=> Summarize work performed from July 1, 1992 through June 30, 1993 for NASA grant NAG5-2006.

=> Discuss research directions for July 1, 1993 through June 30, 1994 which will include a study of integrated compressor/coder system performance for LANDSAT VII.

OUTLINE

*** I. Work Summary for 1992-1993**

- A. Overview of the Communications Link and ANalysis simulation tool (CLEAN)
- B. Error analysis results for the White Sands Ku-band downlink test data acquired from GSFC
- C. Present additional theoretical code performance results on burst errors
- D. Summarize conclusions
- E. Identify publications resulting from the research

II. Work Directions for 1993-1994

- A. Build or acquire and integrate RICE compression/decompression algorithms into CLEAN
- B. Acquire satellite data of RICE compressed/decompressed data
- C. Study coding/compression performance with the acquired satellite data.
- D. Draw conclusions about achievable performance

III. RICE Compression

- A. Data Compression versus Data Expansion
- B. Performance measures
- C. Compression Scheme Issues
- D. The RICE compression algorithm
- E. Pitfalls to avoid

IV. Compression/FEC Integration

- A. Issues
- B. Analysis Approach (Simulation)
- C. Design Approach

OVERVIEW OF CLEAN

=> CLEAN is a simulation tool to investigate performance of codes in random and non-random error environments.

=> Includes:

-> Random and non-random error sequence generation

* Burst length distribution can be fixed, Gaussian, and Poisson
* Burst interval distribution can be periodic, Gaussian, and Poisson
* Error density within the bursts and outside the bursts is user selectable

-> Decoding Operations

* Reed-Solomon ("black box effect")
* BCH (the block decoder which implements the Reed-Solomon code is completely general)
* Viterbi (fully implemented Viterbi decoding algorithm)

-> Interleavers

* Block Interleaver
* Periodic Convolutional Interleaver

-> Error Sequence Analysis Programs

* Bursty-Error parameter estimation (DeltaEst,GAPEst)
* Error distribution tests (Cramer Von-Mises)
* Error Interval Histogram
* Moving Average (MA) filter

-> Utilities/Theoretical function generation

* EOS conversion
* Theoretical Error Interval for random errors
* Many others

WHITE SANDS DATA RESULTS

The White Sands data
received from GSFC
indicates that the errors in the Ku-band downlink
are

RANDOM[*]

[*] A 60Hz fluctuation in the error density was found in some of the data sets.

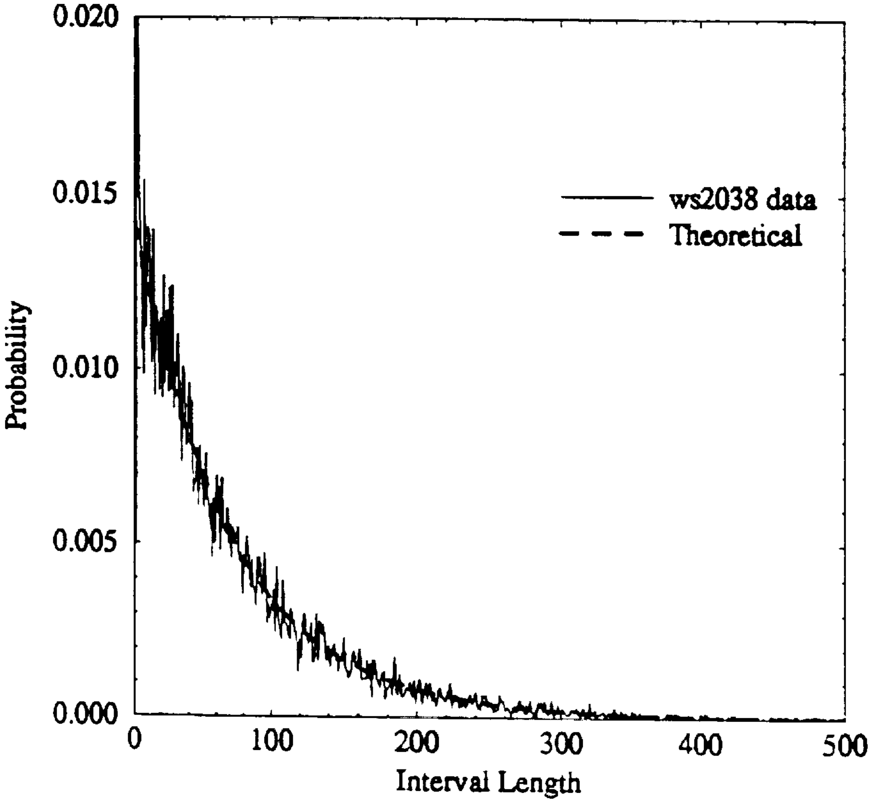


Figure 2. Comparison of the double-error interval distribution for file wsfler642038.dat with the theoretical distribution for Binomial errors.

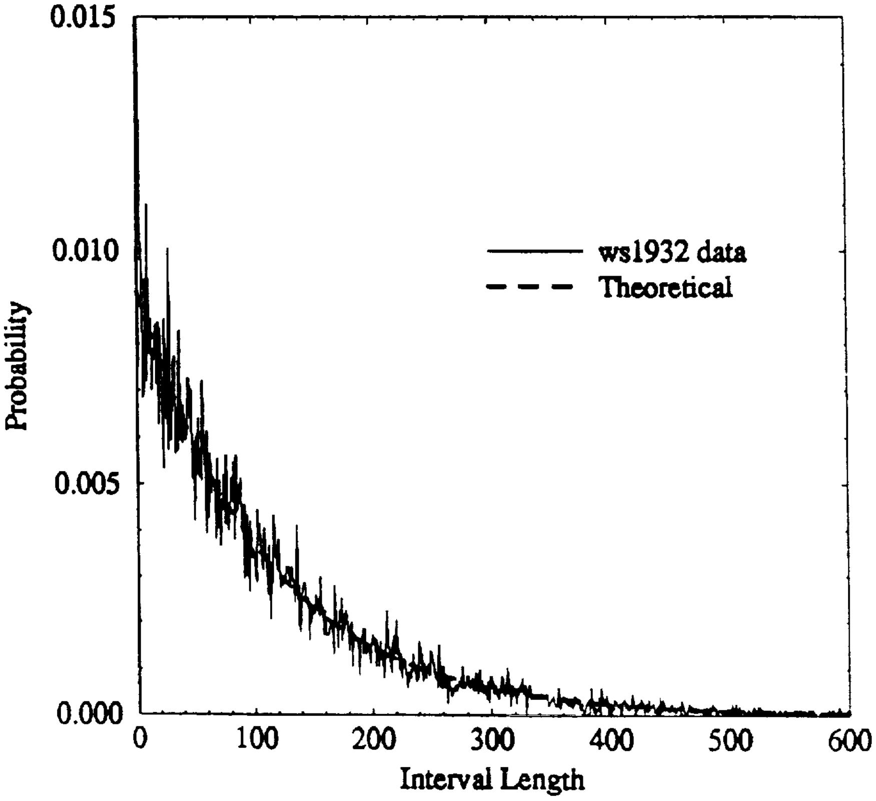
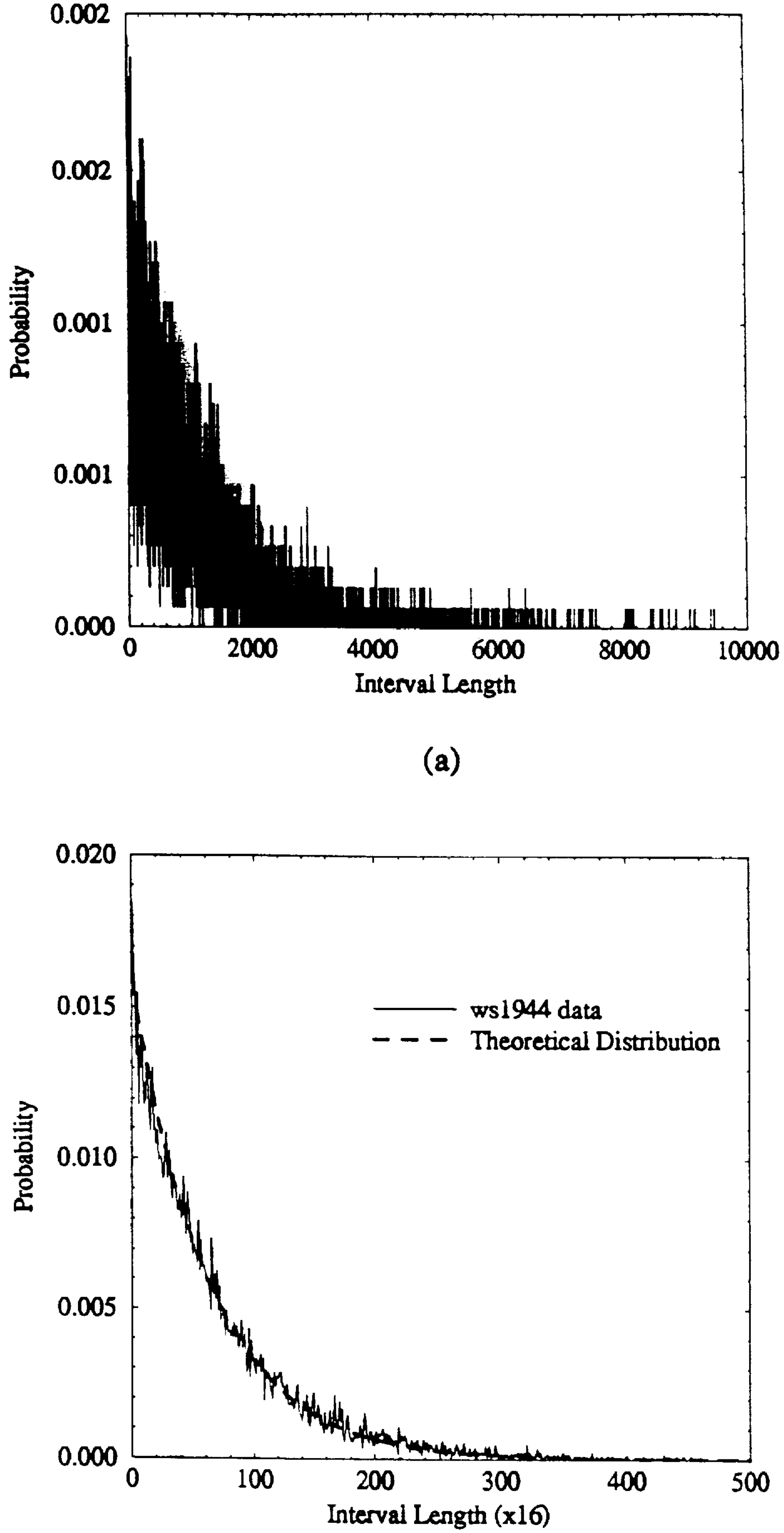


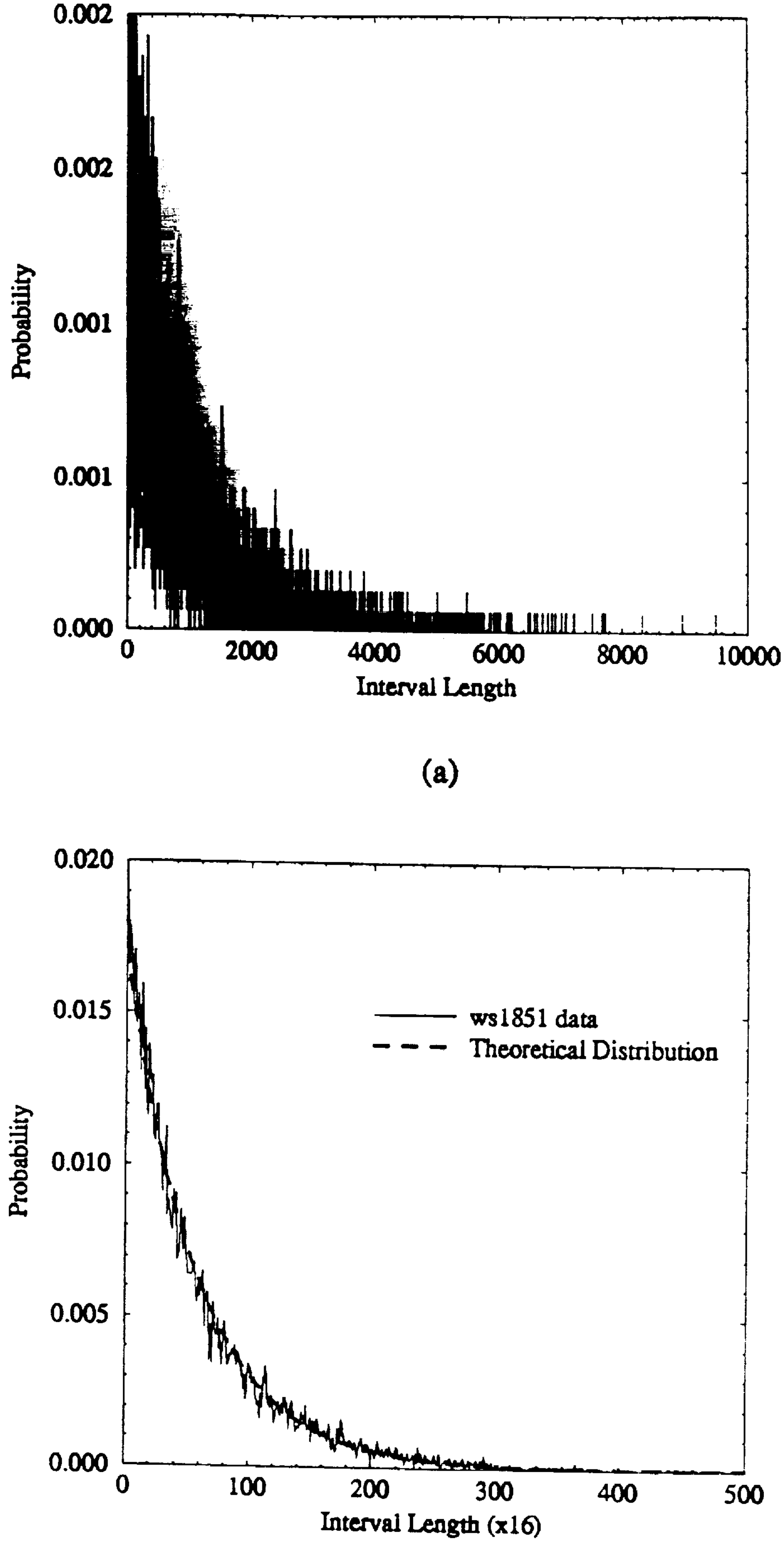
Figure 3. Comparison of the double-error interval distribution for file wsfler641932.dat with the theoretical distribution for Binomial errors.



(a)

(b)

Figure 4. Shown for file wser641944.dat are a) the empirical double-error interval distribution, and b) a comparison of the filtered empirical distribution with the filtered theoretical Binomial-error distribution.



(a)

(b)

Figure 5. Shown for file wser641851.dat are a) the empirical double-error interval distribution, and b) a comparison of the filtered empirical distribution with the filtered theoretical Binomial-error distribution.

CRAMER VON-MISES (CVM) TEST STATISTICS

File No.	Original Name	Error Sequence Length	Number of Errors*	CVM Test Statistic
1	wser641621.dat	2,087,325	12	0.3684
3	wser641716.dat	2,087,325	16	0.1457
5	wser641837.dat	2,087,325	412	0.1609
8	wser641908.dat	2,087,325	38	0.0433
13	wsfler641932.dat	2,087,325	37184	2.4308

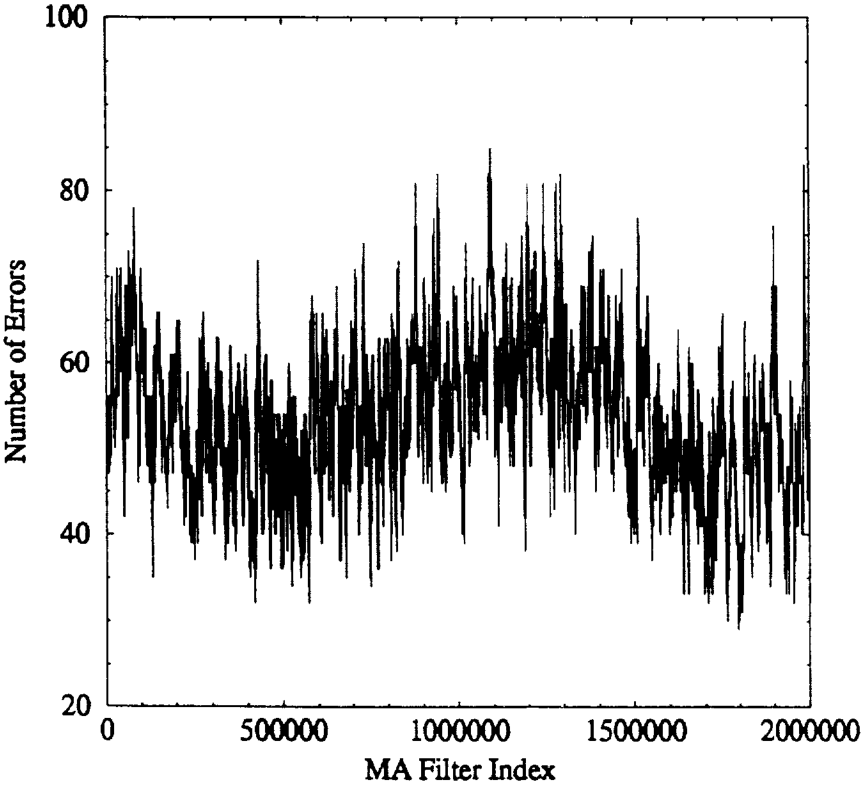
α	Critical Value
0.1	0.347
0.05	0.4610
0.025	0.5810
0.01	0.743
0.001	1.168

$$(80\,\text{Mbps})^{-1}\left(1{,}300{,}000\,\frac{\text{bits}}{\text{cycle}}\right) = .01625\ \text{Sec/Cycl}$$

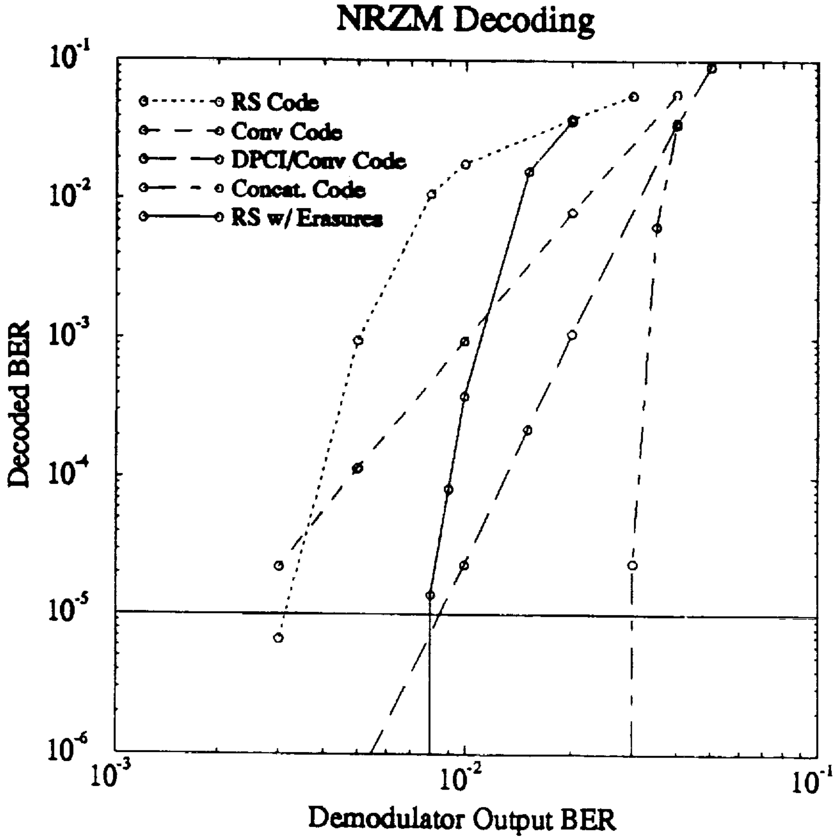
$$\Rightarrow 61.5\ \text{HZ!}$$

Moving Average Filter of Error Sequence

WS1932, at NRZM decoder input, Window=6000, Shift=200

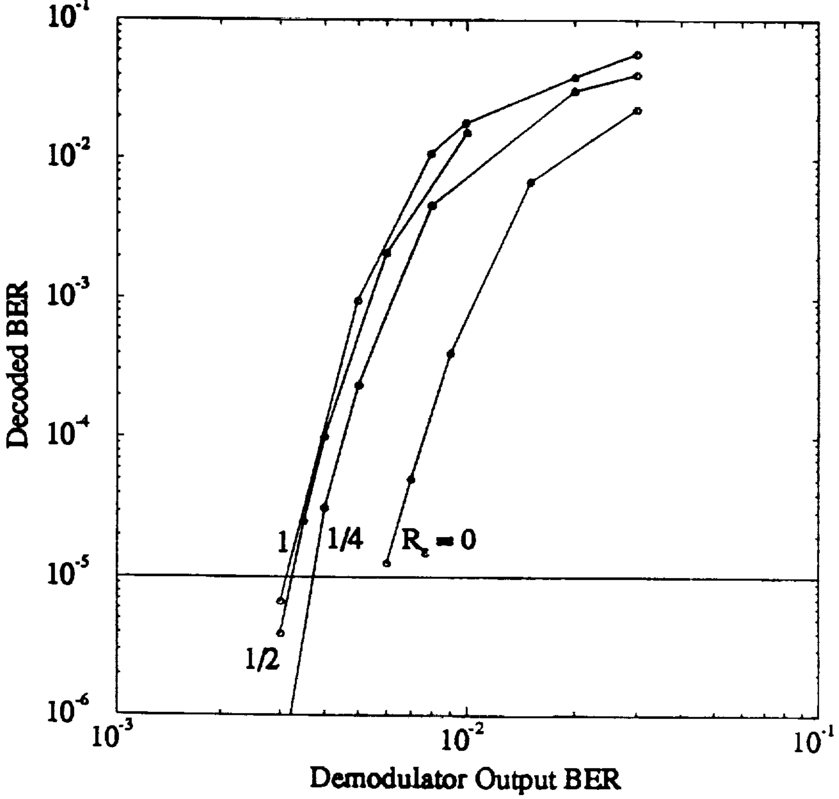


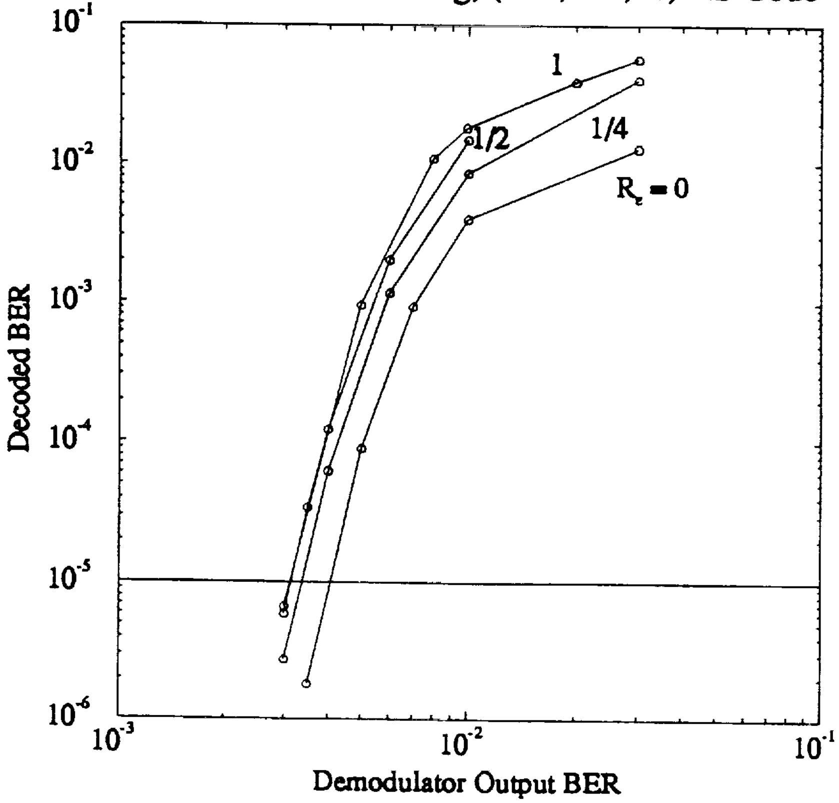
Random-Error Channel Performance



Bursty-Error Channel Performance

d=15, NRZM Decoding, (255,223,16) RS Code



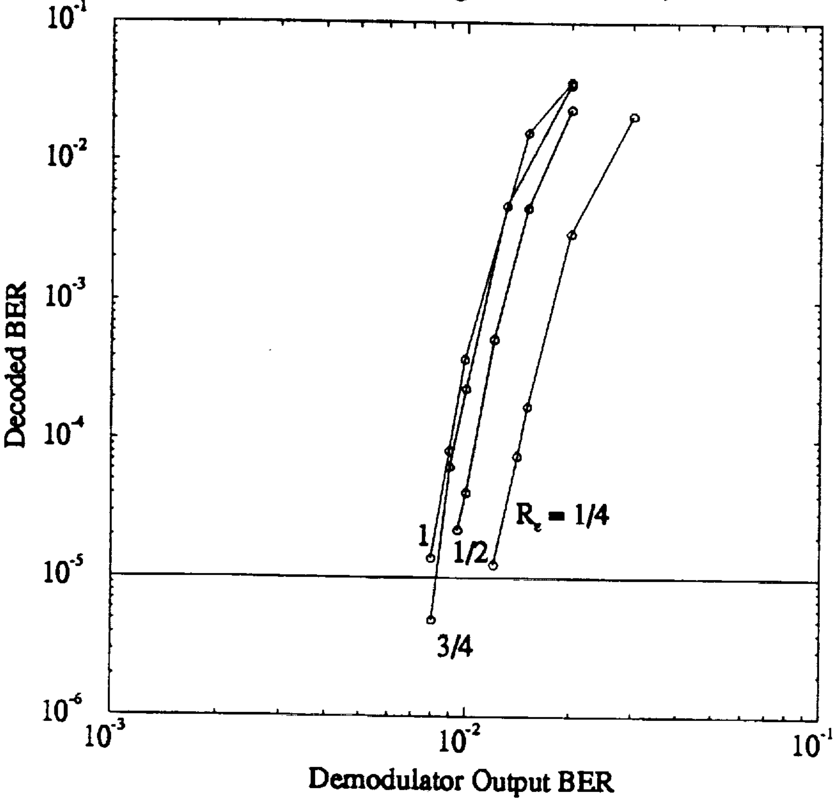


Bursty-Error Channel Performance

d=50, NRZM Decoding, (255,223,16) RS Code

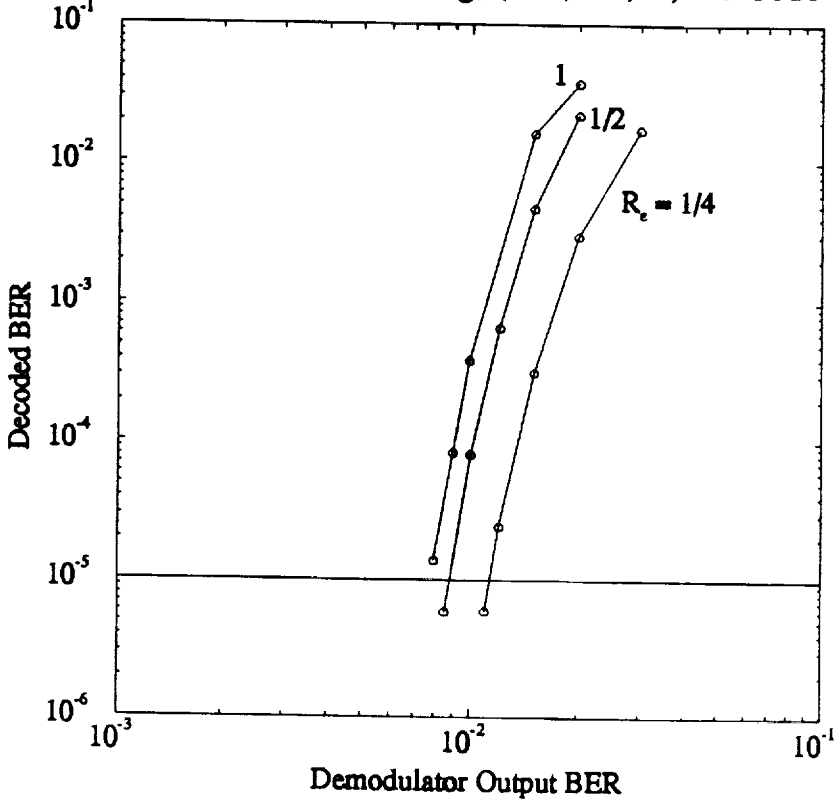
Bursty-Error Channel Performance

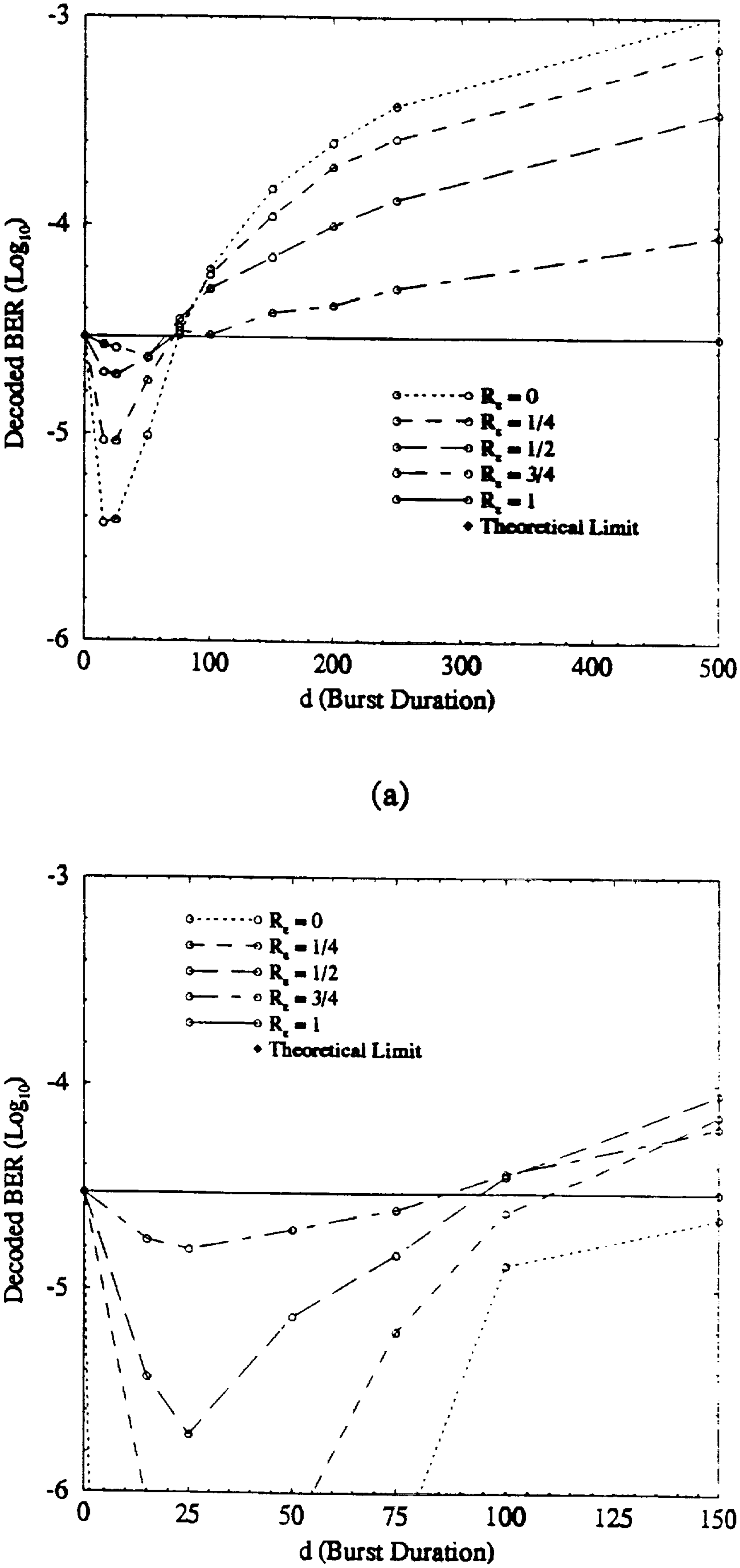
d=25, NRZM Decoding, (255,223,32) RS Code



Bursty-Error Channel Performance

d=50, NRZM Decoding, (255,223,32) RS Code





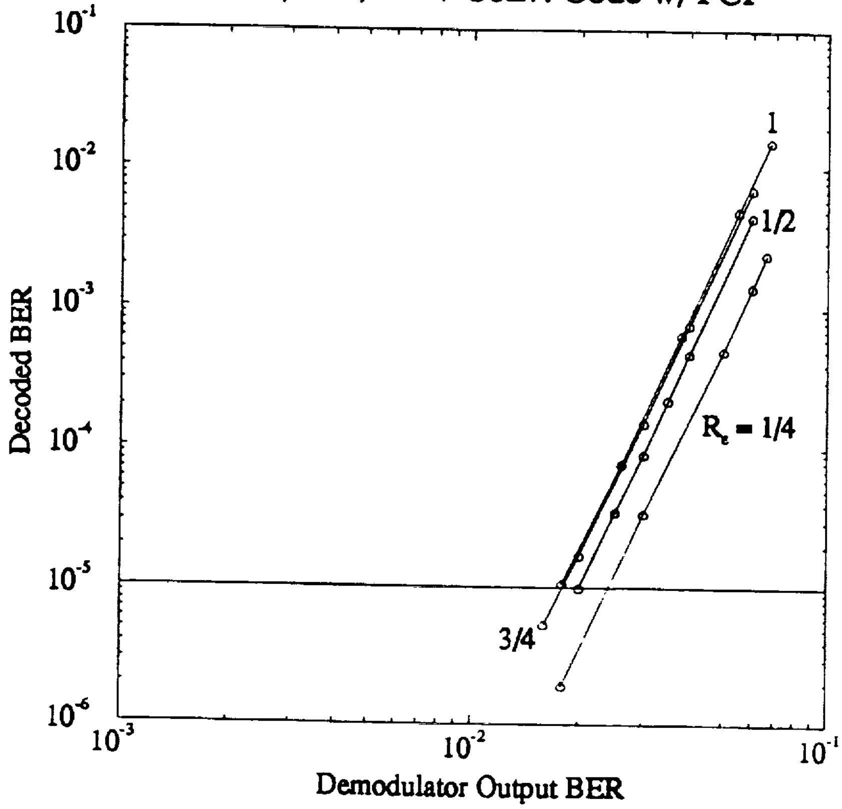
(a)

(b)

Figure 8. Performance of a (255,223) Reed-Solomon code with a depth 5 block interleaver operating on bursty errors with $P_{ch} = 0.004$ and with (a) $\nu d = 0.05$, and (b) $\nu d = 0.01$.

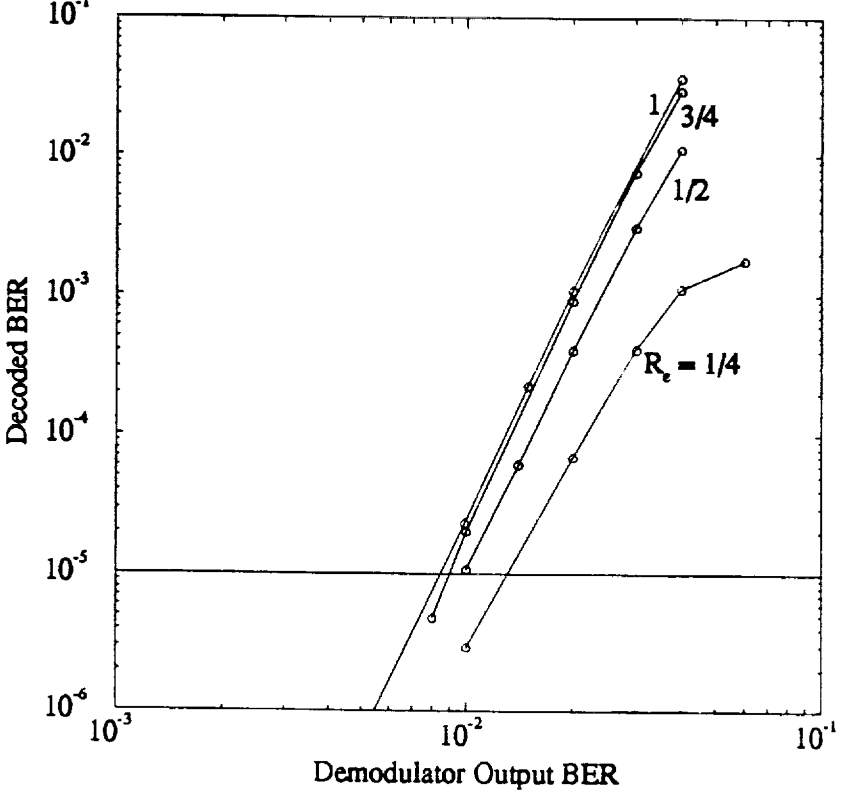
Bursty-Error Channel Performance

d=15, R=1/2 k=7 Conv. Code w/ PCI



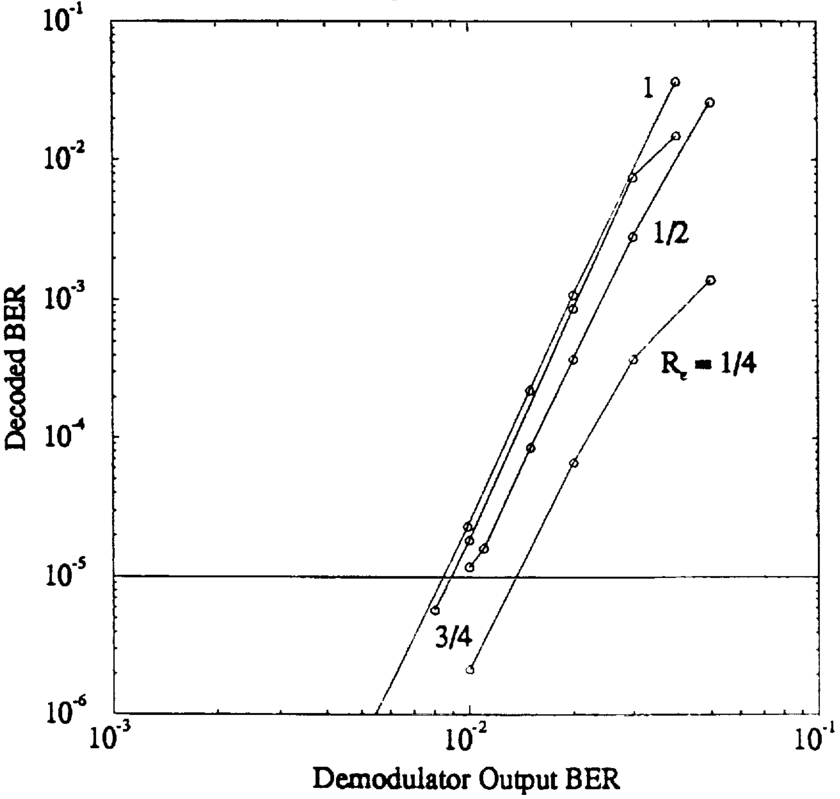
Bursty-Error Channel Performance

d=15, NRZM Decoding, R=1/2 k=7 Conv. Code w/ PCI



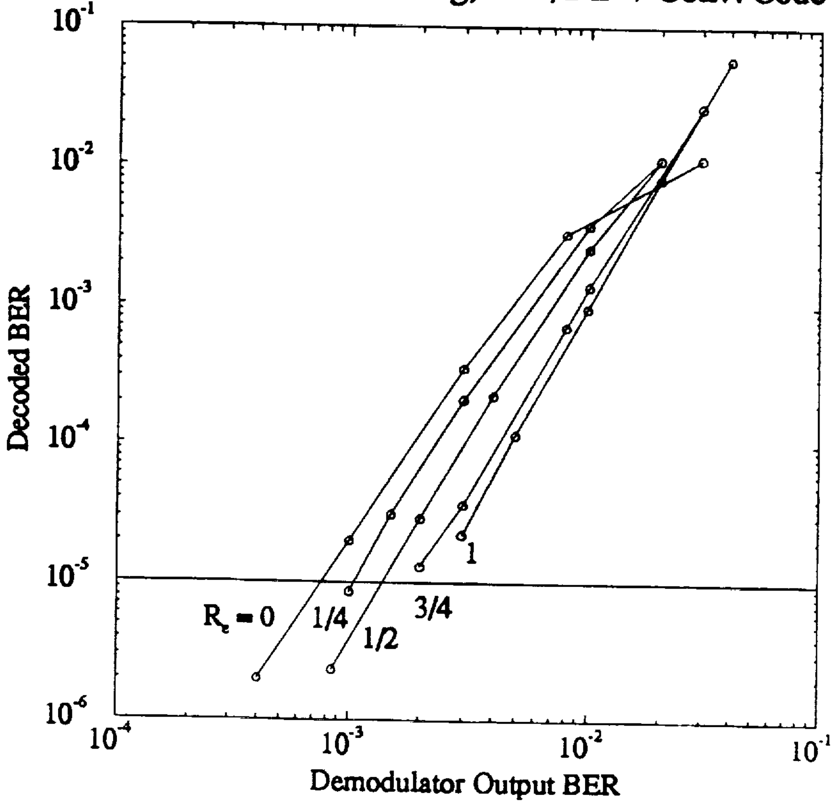
Bursty-Error Channel Performance

d=25, NRZM Decoding, R=1/2 k=7 Conv. Code w/ PCI



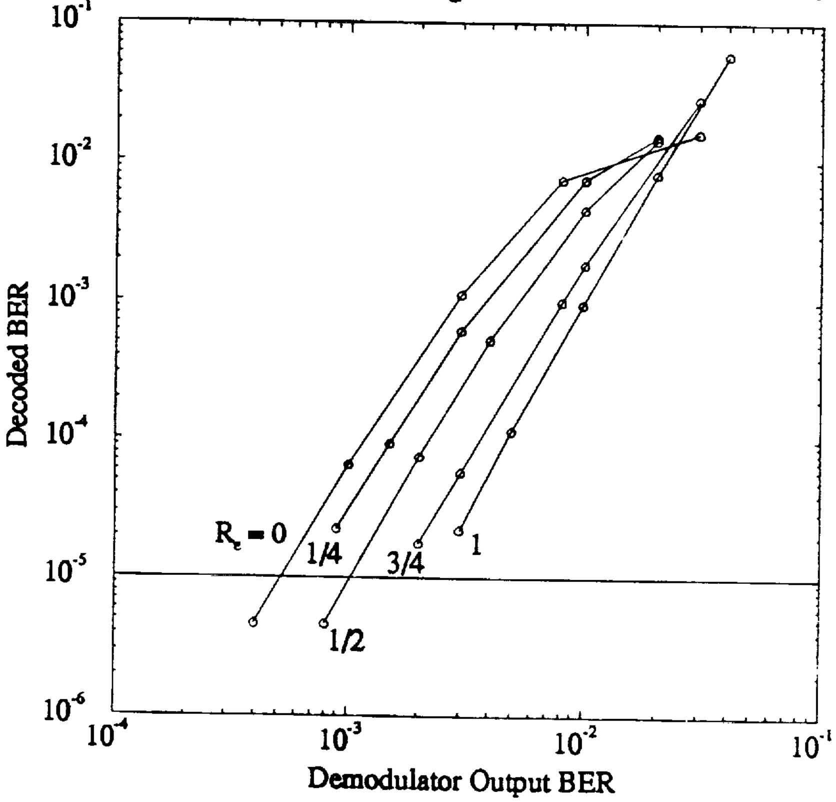
Bursty-Error Channel Performance

d=15, NRZM Decoding, R=1/2 k=7 Conv. Code



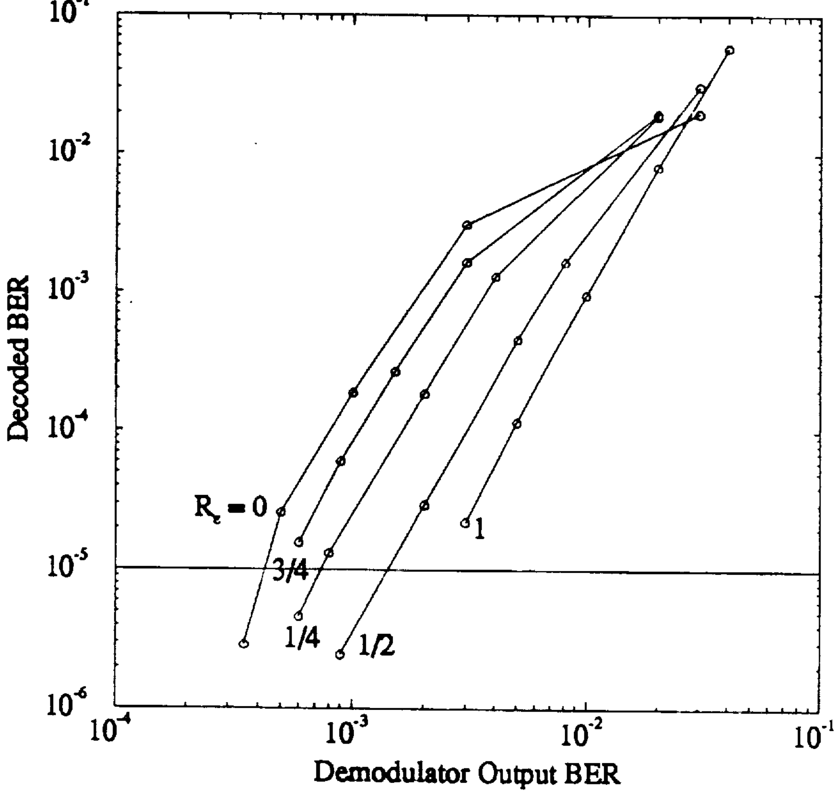
Bursty-Error Channel Performance

d=25, NRZM Decoding, R=1/2 k=7 Conv. Code



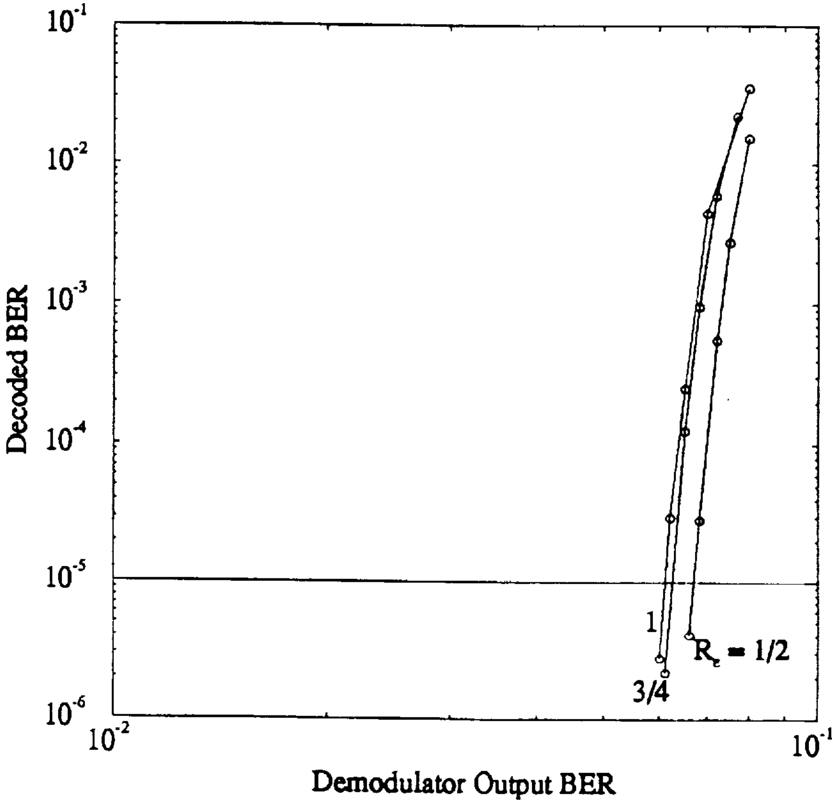
Bursty-Error Channel Performance

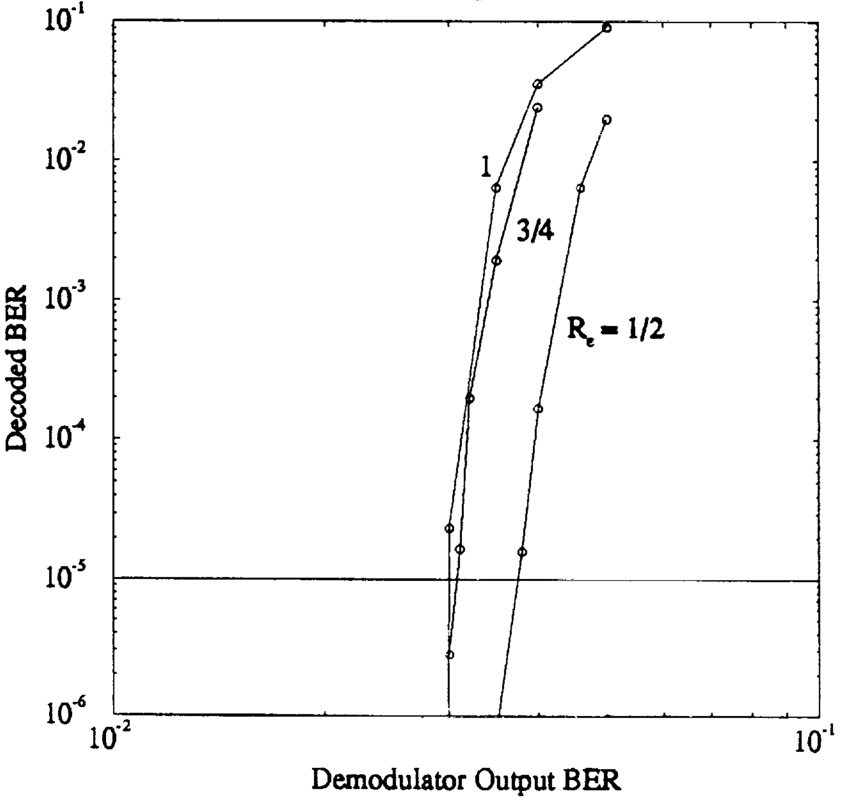
d=50, NRZM Decoding, R=1/2 k=7 Conv. Code



Bursty-Error Channel Performance

d=15, Concatenated Code



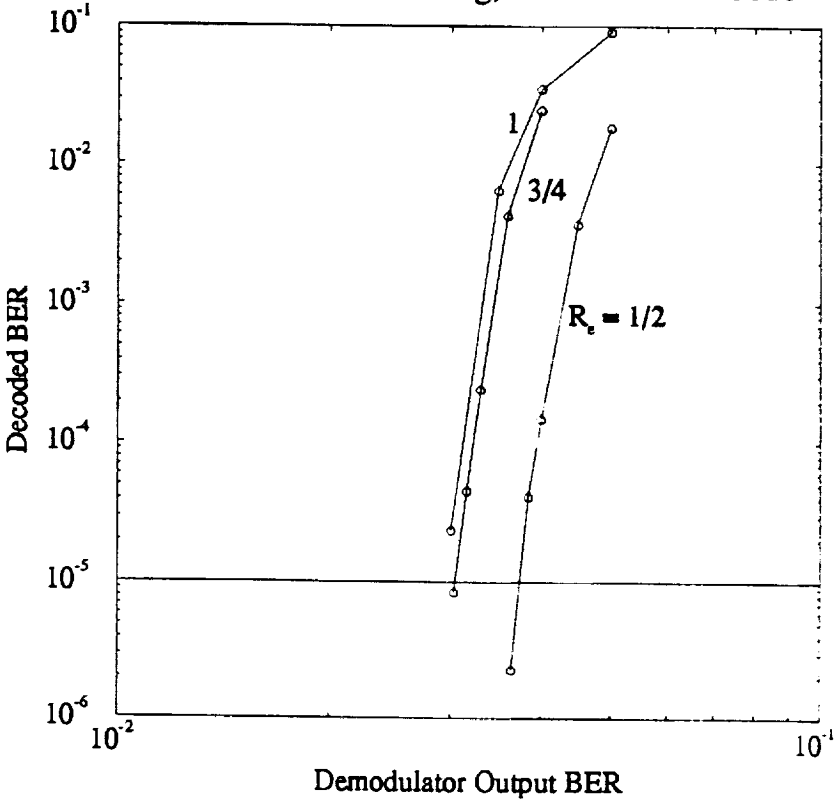


Bursty-Error Channel Performance

d=15, NRZM Decoding, Concatenated Code

Bursty-Error Channel Performance

d=50, NRZM Decoding, Concatenated Code



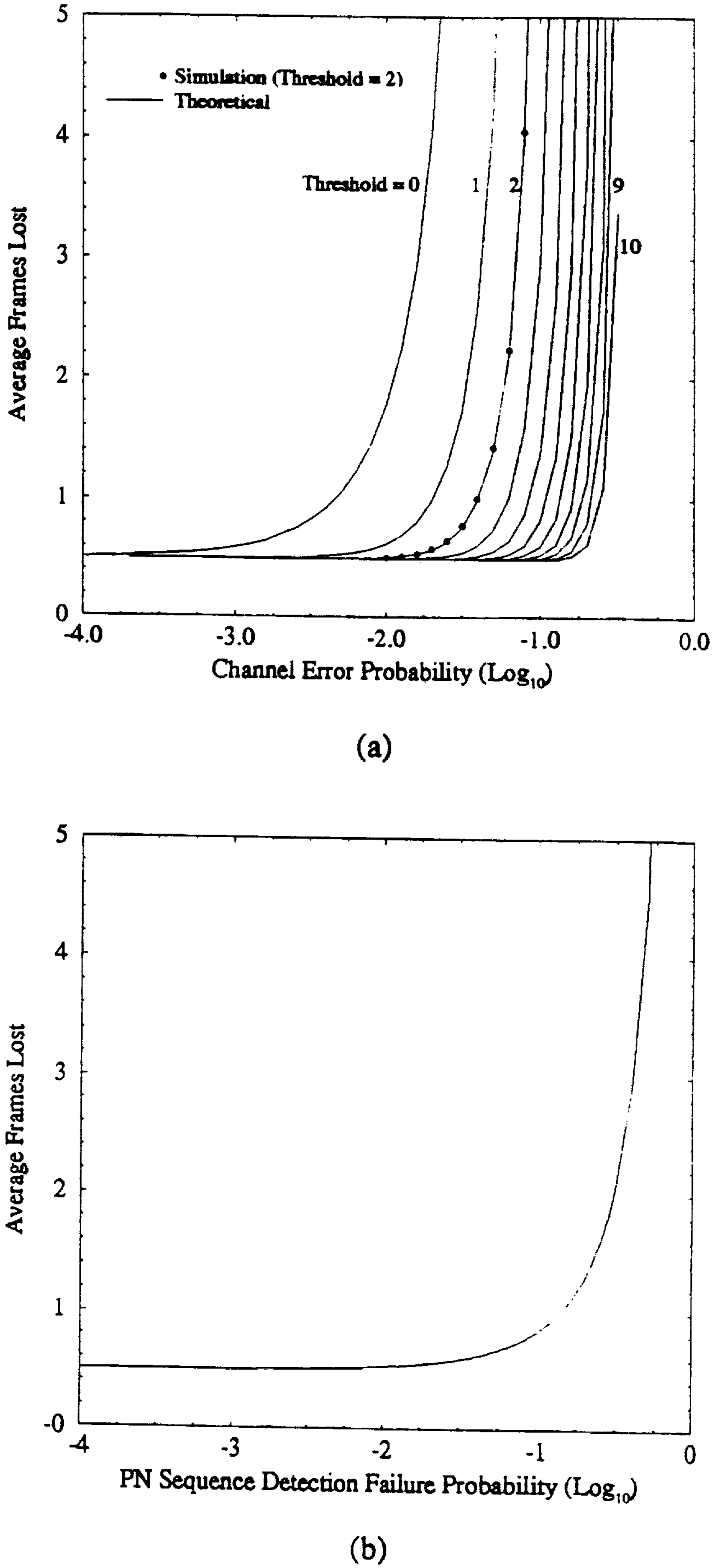
CONCLUSIONS

=> The errors in the real EOS Ku-band downlink data are random

=> Reed-Solomon codes are efficient at correcting bursty errors (performance improves as the error become clustered).

=> Reed-Solomon erasure decoding can significantly improve performance

=> In general, performance degrades as error become clustered for convolutional codes and Viterbi decoding



(a)

(b)

Figure 10. Average frames lost as a function of a) channel error probability assuming independent channel errors, and b) PN sequence header detection failure probability.

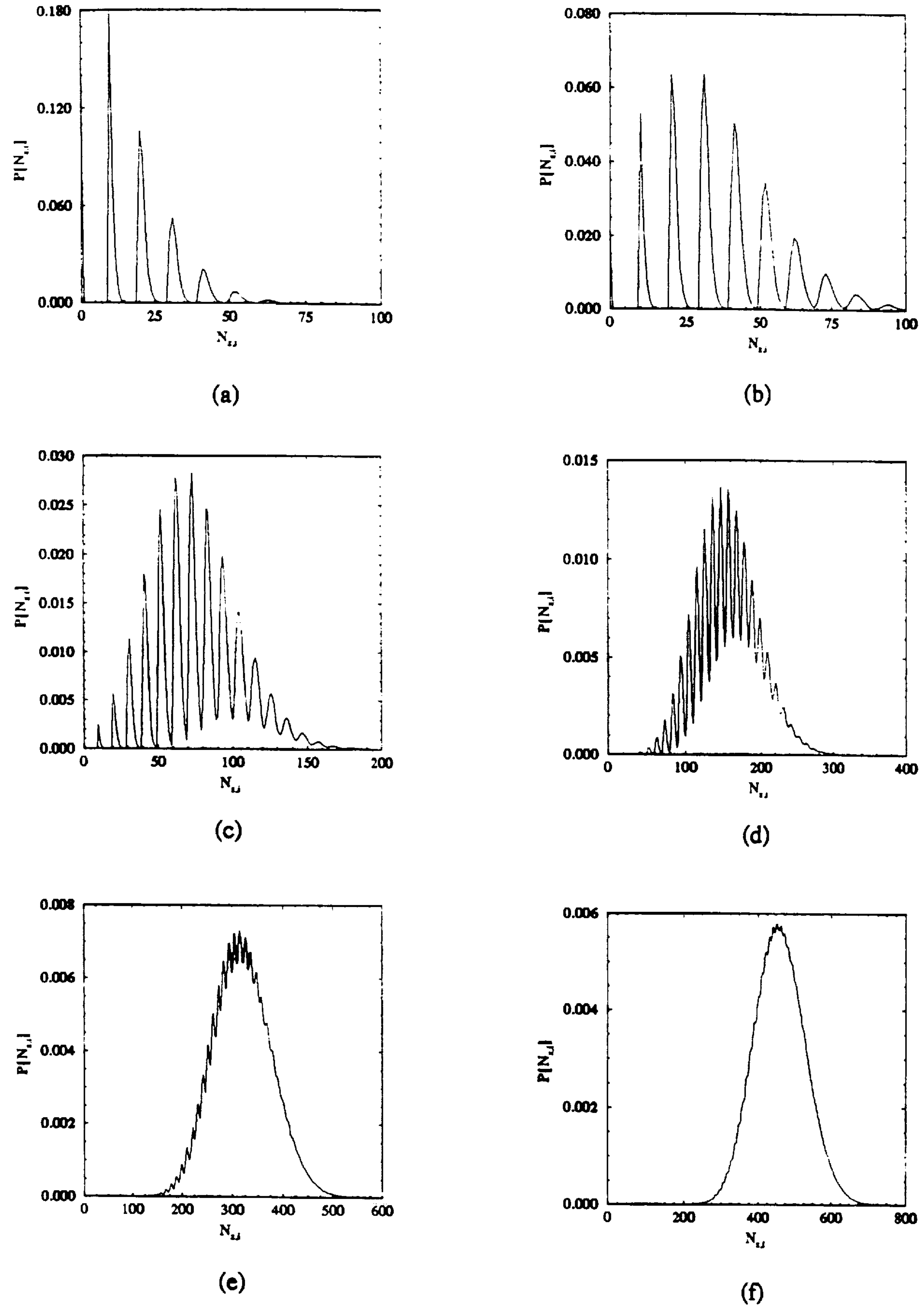


Figure 4. Distribution of binary symbol errors at the decoder output for a (127,109) RS code and for (a) $z = 652$, (b) $z = 1350$, (c) $z = 2610$, (d) $z = 5220$, (e) $z = 10440$, and (f) $z = 15000$.

PUBLICATIONS

=> "Confidence Intervals for Simulations Using Reed-Solomon Codes", W.J. Ebel, F.M. Ingels, MILCOM '93, October 1993, accepted.

=> "Frame Synchronization for the NASA CCSDS Packet Telemetry Standard", W.J. Ebel, IEEE Transactions on Communications, to be submitted.

OUTLINE

I. Work Summary for 1992-1993

- A. Overview of the Communications Link and ANalysis simulation tool (CLEAN)
- B. Error analysis results for the White Sands Ku-band downlink test data acquired from GSFC
- C. Present additional theoretical code performance results on burst errors
- D. Summarize conclusions
- E. Identify publications resulting from the research

*** II. Work Directions for 1993-1994**

- A. Build or acquire and integrate RICE compression/de-compression algorithms into CLEAN
- B. Acquire satellite data of RICE compressed/decompressed data
- C. Study coding/compression performance with the acquired satellite data.
- D. Draw conclusions about achievable performance

III. RICE Compression

- A. Data Compression versus Data Expansion
- B. Performance measures
- C. Compression Scheme Issues
- D. The RICE compression algorithm
- E. Pitfalls to avoid

IV. Compression/FEC Integration

- A. Issues
- B. Analysis Approach (Simulation)
- C. Design Approach

GOALS

=> Integrate RICE compression/decompression algorithms into CLEAN

=> Acquire satellite data of RICE compressed/decompressed data from GSFC

=> Study BCH coding/compression performance with the acquired data

=> Investigate possible alternative coding/compression strategies which might prove more efficient

=> Draw conclusions about achievable performance with the current proposed system and alternative system configurations

OUTLINE

I. Work Summary for 1992-1993

   A. Overview of the Communications Link and ANalysis simulation tool (CLEAN)

   B. Error analysis results for the White Sands Ku-band downlink test data acquired from GSFC

   C. Present additional theoretical code performance results on burst errors

   D. Summarize conclusions

   E. Identify publications resulting from the research

II. Work Directions for 1993-1994

   A. Build or acquire and integrate RICE compression/decompression algorithms into CLEAN

   B. Acquire satellite data of RICE compressed/decompressed data

   C. Study coding/compression performance with the acquired satellite data.

   D. Draw conclusions about achievable performance

*** III. RICE Compression**

   A. Data Compression versus Data Expansion

   B. Performance measures

   C. Compression Scheme Issues

   D. The RICE compression algorithm

   E. Pitfalls to avoid

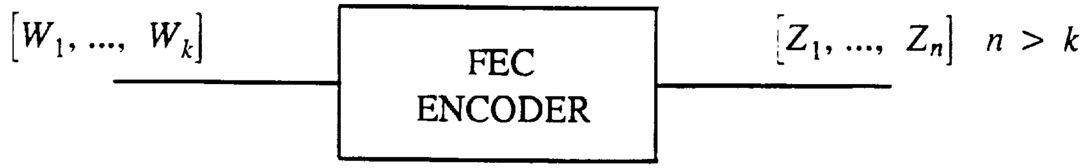
IV. Compression/FEC Integration

   A. Issues

   B. Analysis Approach (Simulation)

   C. Design Approach

DATA COMPRESSION vs. DATA EXPANSION



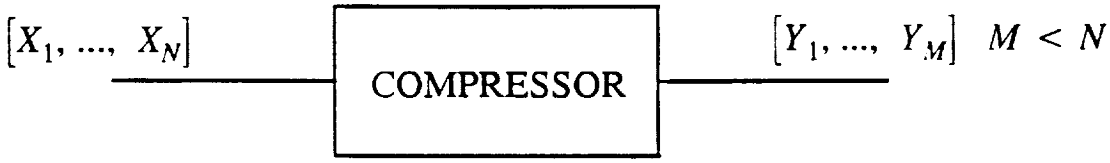
$[W_1, \ldots, W_k]$

FEC ENCODER

$[Z_1, \ldots, Z_n]$ $n > k$

Expansion Factor = n/k

=> **What is compression?** An *algorithm* which eliminates redundancy in a data set so that the # of bits required to represent that data set is minimized.



$[X_1, \ldots, X_N]$

COMPRESSOR

$[Y_1, \ldots, Y_M]$ $M < N$

Compression Ratio = (N/M):1

=> **What types are there?**

-> Lossless (Reversible Operation)

Huffman
RICE
Run Length Coding
White Block Skipping
Shift Code

-> Lossy (Not a Reversible Operation)

Transform
Vector Quantization

PERFORMANCE MEASURES/ CONCEPT ILLUSTRATION

=> **How can we measure the performance of a compression algorithm?** Use the ***entropy*** of the data source as a figure of merit. If a compression algorithm performs near the entropy of the source, we conclude that the algorithm is good.

=> The ***entropy*** of a data source is a measure of the randomness of the source output. It's numerical value is the average number of bits required to represent each source symbol output.

=> For example:

Source Output	Binary Code	Output Prob.	Huffman Code
X_0	00	.5	0
X_1	01	.25	10
X_2	10	.125	110
X_3	11	.125	111
Ave. Bit Length	2 b/sym		1.375 b/sym

The ***entropy***, $H(x)$, of this source is 1.375 bits/source symbol

COMPRESSION SCHEME ISSUES

=> Is lossless compression required, or can some data resolution be lost?

=> What compression ratios are achievable for the various compression algorithms given **YOUR** data? Compression ratios are data dependent.

=> How will errors in the compressed data affect errors in the decompressed data? Is it possible for ***error propagation*** to occur?

=> What are the source data characteristics?
Has the data been companded via a log function before compression?
If so, how will this affect the error statistics after decompression?

=> How will variable (unpredictable) blocklengths of compressed data affect multiplexing/demultiplexing operations? Will we have problems finding the block boundaries after demultiplexing?

=> How can synchronization be achieved to locate compressed data boundaries?
How might errors in the compressed data affect boundary detection?
How might errors in the compressed data affect ID or frame detection?
Note: Boundaries in the compressed data may exist to provide ***error truncation.***

=> How fast can boundaries in the compressed data be detected given the error probability output from the FEC decoder? How much data will be lost before synchronization will occur?

RICE COMPRESSION

* => ***Assumptions:***

-> Source data has been preprocessed so that data samples entering the compressor are independent.

-> Data samples entering the compressor have been relabeled (renumbered) so that higher probability samples have smaller numbers.

-> Parameters of each compression algorithm have been optimized for the processed data statistics anticipated.

=> ***Architecture:***

-> Operates on blocks of processed source data. Block sizes may be determined *a priori* or may be determined on the fly.

-> The compression algorithm is composed of several simple compression algorithms, each of which is designed to efficiently compress a subset of the total span of possible data statistics.

⊙ $\psi_0[\tilde{X}] = C\overline{FS}[\tilde{X}]$

⊙ $\psi_1[\tilde{X}] = FS[\tilde{X}]$ (Golomb prefix code)

⊙ $\psi_2[\tilde{X}] = CFS[\tilde{X}]$

⊙ $\psi_3[\tilde{X}] = \tilde{X}$

⊙ $\psi_4[\tilde{X}] = BC[\tilde{X}] = ID * \psi_{ID}[\tilde{X}]$

-> Simple processing of the current processed data block determines, with high probability, which algorithm will perform the best (near the entropy of the data).

=> ***Operation:***

-> Some processed data is partitioned into a data block.

-> A simplified algorithm is applied to the data block to determine which simple compressor algorithm will perform the best.

-> The data block is compressed with that compressor.

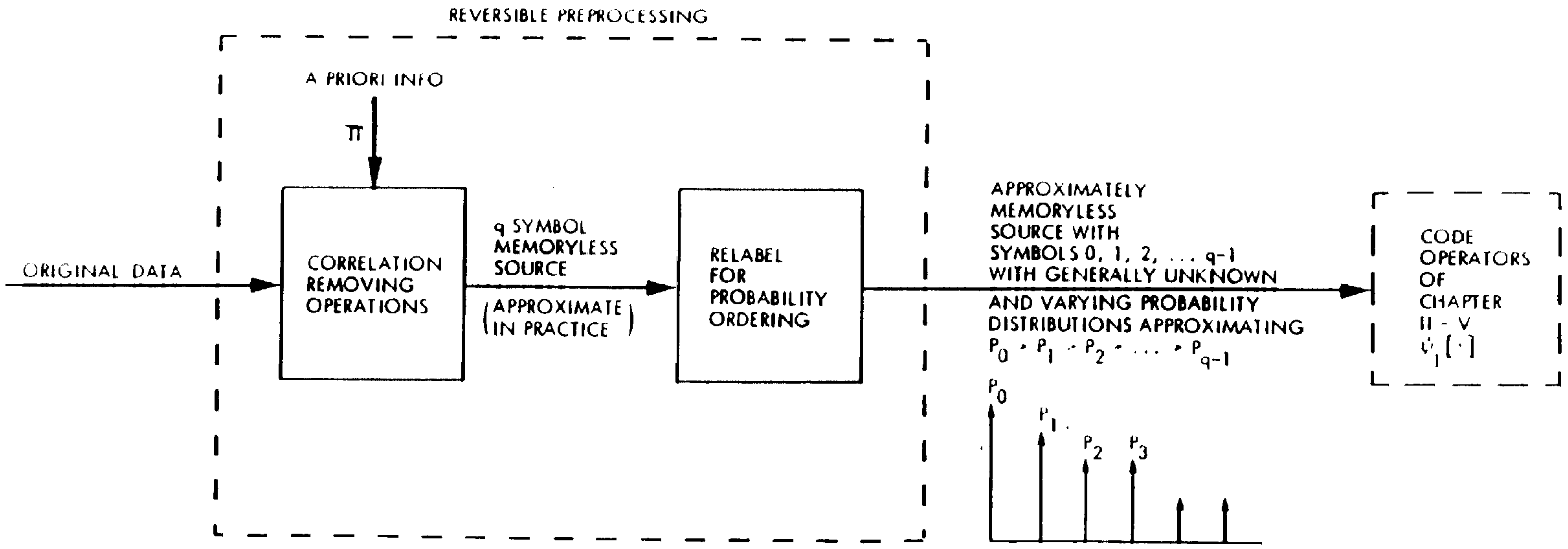
-> The compressor outputs the total block of compressed which includes the algorithm ID plus the compressed data ($\psi_4[\cdot]$).

=> ***Notes:***

-> The RICE compression algorithm is reversible (lossless). ***This does not mean that the decompressed data will be error free!***

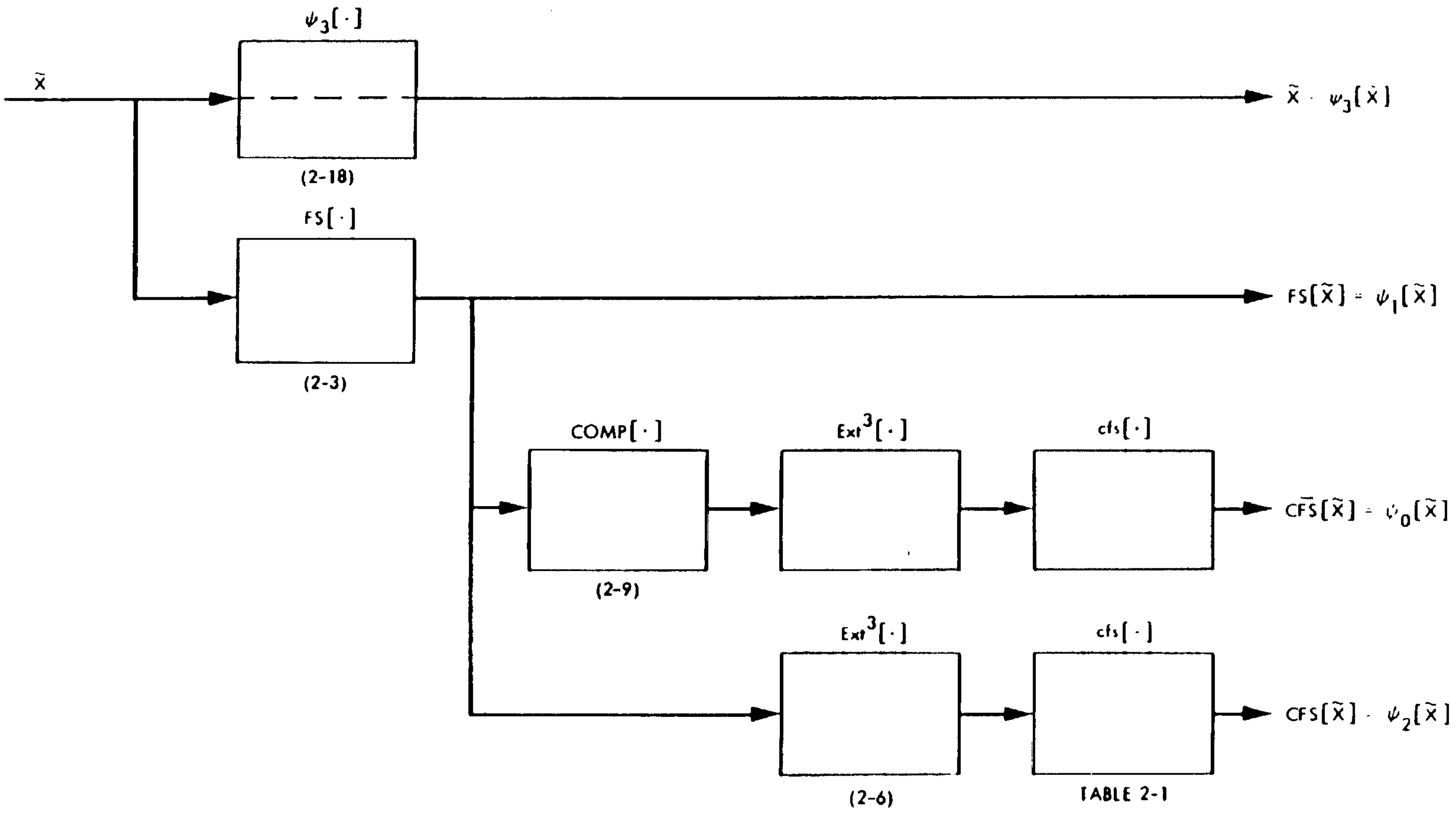
-> If the data statistics during operation are not as expected, ***data expansion can occur!***

REVERSIBLE PREPROCESSING*


REVERSIBLE PREPROCESSING
A PRIORI INFO
π
ORIGINAL DATA
CORRELATION REMOVING OPERATIONS
q SYMBOL MEMORYLESS SOURCE
(APPROXIMATE IN PRACTICE)
RELABEL FOR PROBABILITY ORDERING
APPROXIMATELY MEMORYLESS SOURCE WITH SYMBOLS 0, 1, 2, ... q-1 WITH GENERALLY UNKNOWN AND VARYING PROBABILITY DISTRIBUTIONS APPROXIMATING
$P_0 \geq P_1 \geq P_2 \geq \dots \geq P_{q-1}$
P_0
P_1
P_2
P_3
CODE OPERATORS OF CHAPTER II - V
$\psi_i[\cdot]$

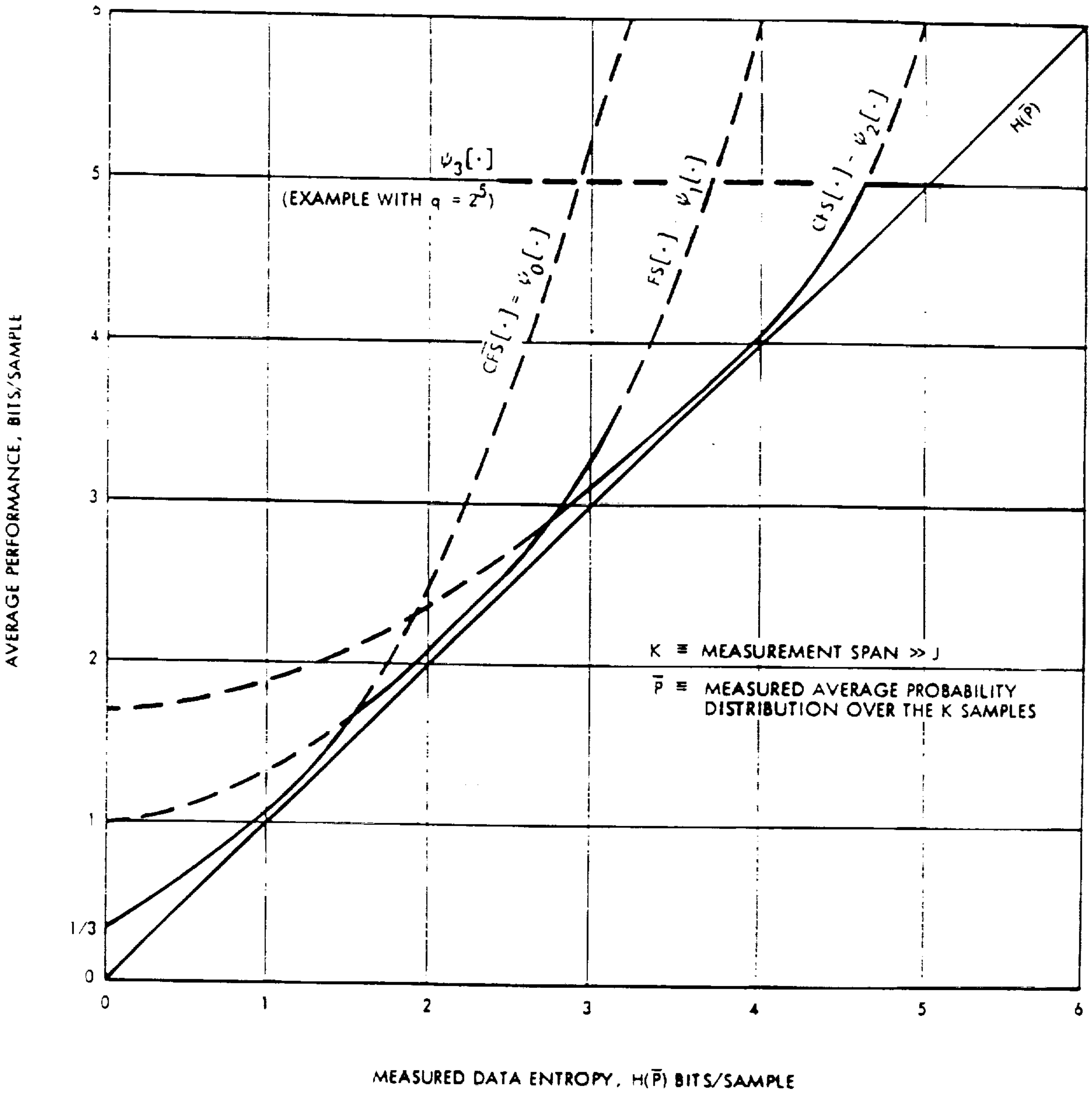
*Taken from Rice, R.F., "Some Practical Universal Noiseless Coding Techniques", JPL Pub. 79-22, p. 7

FOUR SIMPLE COMPRESSION ALGORITHMS*


$\tilde{x}$
$\psi_3[\cdot]$
(2-18)
$\tilde{x} = \psi_3[\tilde{x}]$
$FS[\cdot]$
(2-3)
$FS[\tilde{x}] = \psi_1[\tilde{x}]$
$COMP[\cdot]$
(2-9)
$Ext^3[\cdot]$
$cfs[\cdot]$
$C\overline{FS}[\tilde{x}] = \psi_0[\tilde{x}]$
$Ext^3[\cdot]$
(2-6)
$cfs[\cdot]$
TABLE 2-1
$CFS[\tilde{x}] = \psi_2[\tilde{x}]$

*Taken from Rice, R.F., "Some Practical Universal Noiseless Coding Techniques", JPL Pub. 79-22, p. 7

AVERAGE PERFORMANCE OF EACH SIMPLE COMPRESSOR*



*Taken from Rice, R.F., "Some Practical Universal Noiseless Coding Techniques", JPL Pub. 79-22, p. 7

PITFALLS TO AVOID

=> The RICE algorithm partitions the original data into groups of 16 samples (optimum for a specific set of data). Is this optimum for ***YOUR*** data? If not, the compression ratios for your data might be intolerably low!

=> The RICE algorithm assumes that the original source data has been preprocessed so that samples are ***independent***. Is true for the data input to the Lossless Image Compression Chip Set? YES! Has your data been successfully preprocessed? If not, the compression ratios for your data might be intolerably low! How can one determine if the samples are sufficiently independent?

=> The RICE algorithm assumes that, even though the exact distribution for the preprocessed data samples is unknown, the ***ORDERING*** of the probabilities of the preprocessed data samples is roughly known. Do you know this ordering for ***YOUR*** data? If not, and you guess wrong, the compression ratios for your data might be intolerably low! What if the probability ordering changes as a function of time? The RICE compression algorithm allows compression near the entropy for the processed data if the entropy of the processed data is within some predetermined bounds. However, even if a block of your data satisfies this entropy range, your compression may be a far cry from the entropy ***IF*** your probability orderings are off base!

=> It is possible to over design the compressor. Suppose that your entropy ranges are $3<H<5$. Then to design a compressor which works on entropy ranges $0<H<8$ is unnecessary and could very well complicate the decompression operation, resulting in higher sensitivity to errors, to the point where an acceptable solution does not appear possible!

=> Does the decompression assume that there are not errors in the compressed data? What happens if there are errors in the compressed data? There are issues dealing with handling inconsistencies in the decoding operation which must be dealt with for proper chip set operation.

=> Good performance of the RICE compression algorithm requires that the expected variations in the data be well understood. It requires that highly redundant portions of the data are carefully exploited. ***EVEN IF YOU HAVE SUCCESSFULLY PREPROCESSED THE SOURCE DATA AND YOU SUCCESSFULLY ORDER, BY RANKING THE PROBABILITIES, THE PROCESSED DATA SAMPLES, THE RICE COMPRESSION ALGORITHM MAY NOT PERFORM WELL!*** As an example, see compressor ψ_7.

OUTLINE

I. Work Summary for 1992-1993

   A. Overview of the Communications Link and ANalysis simulation tool (CLEAN)

   B. Error analysis results for the White Sands Ku-band downlink test data acquired from GSFC

   C. Present additional theoretical code performance results on burst errors

   D. Summarize conclusions

   E. Identify publications resulting from the research

II. Work Directions for 1993-1994

   A. Build or acquire and integrate RICE compression/de-compression algorithms into CLEAN

   B. Acquire satellite data of RICE compressed/decompressed data

   C. Study coding/compression performance with the acquired satellite data.

   D. Draw conclusions about achievable performance

III. RICE Compression

   A. Data Compression versus Data Expansion

   B. Performance measures

   C. Compression Scheme Issues

   D. The RICE compression algorithm

   E. Pitfalls to avoid

*** IV. Compression/FEC Integration**

   A. Issues

   B. Analysis Approach (Simulation)

   C. Design Approach

COMPRESSION/FEC INTEGRATION ISSUES

=> Are both a compressor and FEC required in the system to combat channel errors and still provide the overall required compression?

=> What is the best way to integrate the compressor/FEC/interleaver within a telemetry frame or packet? Stated another way, what telemetry format constraints are placed on the system?

=> How much do the channel statistics vary (assuming they are not stationary) with respect to a compressed block of data?

-> If an FEC is used, can it effectively correct errors resulting from the extremes of the varying channel error statistics?

-> If not, how does one select a compression algorithm and FEC which is compatible with the telemetry frame format and which can also effectively correct the extremes of the statistically varying channel errors?

-> Is it possible that the problem is overly constrained?

=> What FEC and compression algorithm have similar architectures? Which of those architectures are similar to the type of data output by the data source?

ANALYSIS APPROACH

=> *Simulation* is the best approach for the analysis. Why?

-> Performance is strongly dependent upon data/channel statistics which probably cannot be modeled mathematically.

-> The complexity of the overall system will, in all likelihood, not result in tractable mathematics.

-> A simulation, designed to conduct trade studies, could be easily adapted as a lab tool for performing, in software, compression of data.

=> What should the simulation be able to do?

-> Perform the complete compression/decompression operations for selected algorithms

-> Perform "black box" effects of the channel. This could be accomplished using:

- Real Channel error data
- CLASS simulated channel error data
- Simplified Mathematical models

-> Perform "black box" effects of various FECs.

-> Investigate synchronization issues

-> Investigate error propagation

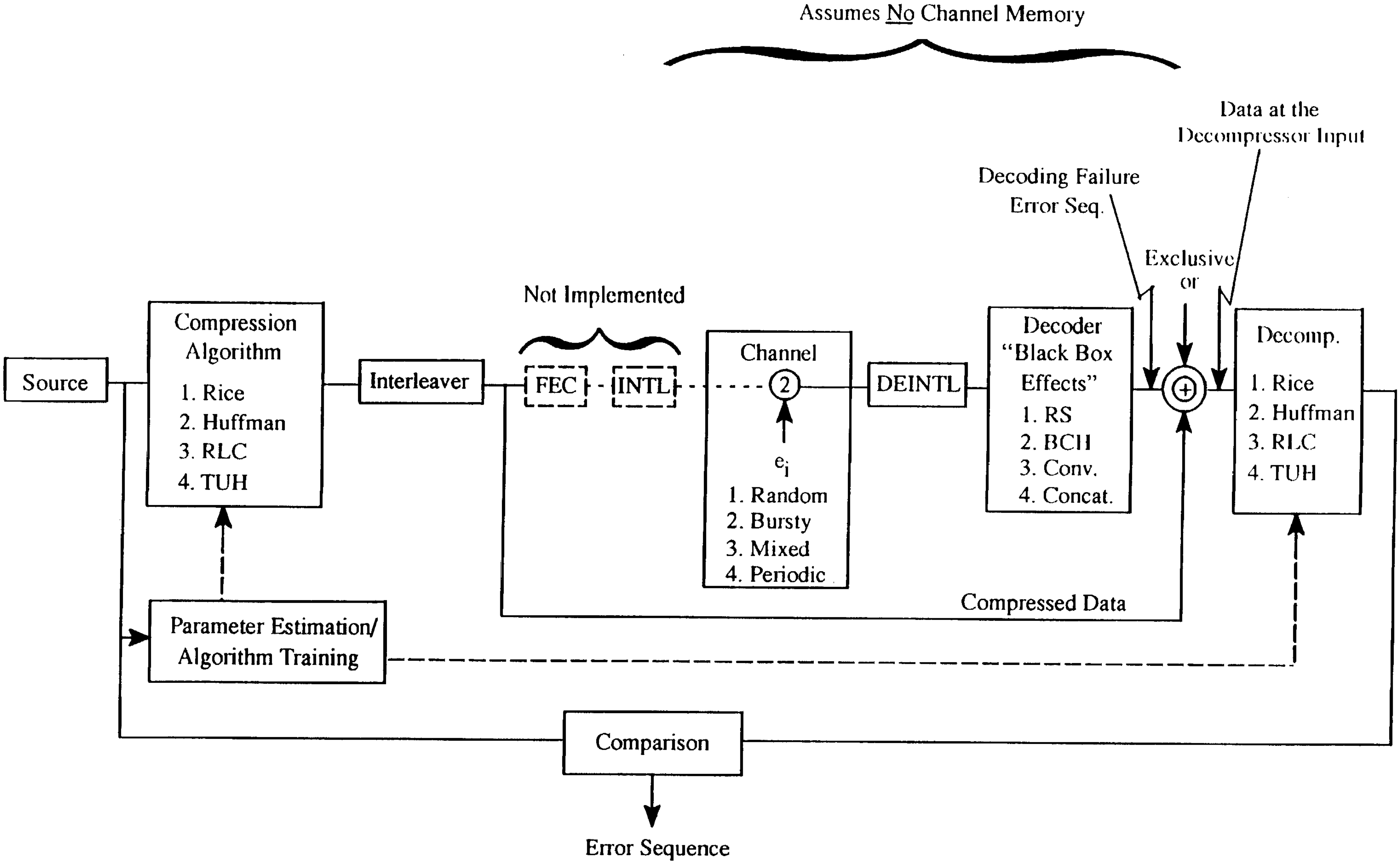


Figure 2. Simulation Block Diagram

DESIGN APPROACH

=> What approach/strategy might a designer use to select a good integrated compressor/FEC scheme taking into account the channel effects?
Try this...

=> **First,** study the compression algorithms in detail.

-> Determine the expected compression ratio with ***your*** data

-> Study the effects on the decompressed data when errors occur on the compressed data.

-> Understand whether and to what extent ***error propagation*** might occur.

=> **Second,** study the FECs/interleavers in detail.

-> Determine the type of errors each FEC is designed to correct.

-> Determine the type and density of errors output by each FEC given the channel error statistics.

=> **Third,** study the channel to determine the type (random, bursty, mixed, periodic) and density of errors expected.

=> **Fourth,** iterate as following:

-> Select a compression scheme so that the total effective compression ratio (compressor compression ratio times the ***expansion factor*** of the FEC) just meets the spec. Note: at first do ***not*** place an FEC in the system design.

-> From studies previously conducted, determine whether errors in the channel will result in acceptable performance for the decompressed data.

-> If yes, the design is complete.
If no, choose an FEC to provide more error protection (this will surely increase the FEC ***expansion factor***).

=> Note: Best design will meet spec while minimizing hardware/algorithm complexity, cost, size, weight, power, ... It might be helpful to choose a compressor and FEC with similar algorithm architecture.

=> Note: It may not be possible to satisfy all the system requirements!